NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF A FAULT TOLERANT COMPUTING TESTBED: A TOOL FOR THE ANALYSIS OF HARDWARE AND SOFTWARE FAULT HANDLING TECHNIQUES

by
David C. Summers

June 2000

Thesis Co-advisors:	Alan A. Ross
	Herschel H. Loomis

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2000	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE Implementation of a Fault Tolerant Computing Testbed: A Tool for the Analysis of Hardware and Software Fault Handling Techniques	**5. FUNDING NUMBERS**
6. AUTHOR(S) Summers, David C.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

With spacecraft designs placing more emphasis on reduced cost, faster design time, and higher performance, it is easy to understand why more commercial-off-the-shelf (COTS) devices are being used in space based applications. The COTS devices offer spacecraft designers shorter design-to-orbit times, lower system costs, orders of magnitude better performance, and a much better software availability than their radiation hardened (radhard) counterparts. The major drawback to using COTS devices in space is their increased susceptibility to the effects of radiation, single event upsets (SEUs) in particular.

This thesis will focus on the implementation of a fault tolerant computer system. The hardware design presented here has two different benefits. First, the system can act as a software testbed, which allows testing of software fault tolerant techniques in the presence of radiation induced SEUs. This allows the testing of the software algorithms in the environment they were designed to operate in without the expense of being placed in orbit. Additionally, the design can be used as a hybrid fault tolerant computer system. By combining the masking ability of the hardware with supporting software, the system can mask out and reset processor errors in real time. The design layout will be presented using OrCAD® schematics.

14. SUBJECT TERMS Fault Tolerant Computing, Triple Modular Redundancy (TMR), Commercial-Off-The-Shelf (COTS) Devices, Single Event Upsets (SEU)	**15. NUMBER OF PAGES**
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

IMPLEMENTATION OF A FAULT TOLERANT COMPUTING TESTBED: A TOOL FOR THE ANALYSIS OF HARDWARE AND SOFTWARE FAULT HANDLING TECHNIQUES

David C. Summers
Captain, United States Marine Corps
B.S., Texas A&M University, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2000

Author: [illegible]
David C. Summers

Approved by: Alan A Ross
Alan A. Ross, Thesis Co-advisor

[illegible]
Herschel H. Loomis, Thesis Co-advisor

[illegible]
Jeffrey B. Knorr, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

With spacecraft designs placing more emphasis on reduced cost, faster design time, and higher performance, it is easy to understand why more commercial-off-the-shelf (COTS) devices are being used in space based applications. The COTS devices offer spacecraft designers shorter design-to-orbit times, lower system costs, orders of magnitude better performance, and a much better software availability than their radiation hardened (radhard) counterparts. The major drawback to using COTS devices in space is their increased susceptibility to the effects of radiation, single event upsets (SEUs) in particular.

This thesis will focus on the implementation of a fault tolerant computer system. The hardware design presented here has two different benefits. First, the system can act as a software testbed, which allows testing of software fault tolerant techniques in the presence of radiation induced SEUs. This allows the testing of the software algorithms in the environment they were designed to operate in without the expense of being placed in orbit. Additionally, the design can be used as a hybrid fault tolerant computer system. By combining the masking ability of the hardware with supporting software, the system can mask out and reset processor errors in real time. The design layout will be presented using OrCAD® schematics.

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to take this opportunity to thank all the people who provided the support and assistance that made this work possible.

First of all, to my family, Sandra, Courtney, and Kristen Summers: Without your unwavering support throughout this endeavor, its completion would not have been possible.

To Captain Kim Whitehouse and Susan Groening: Your software expertise provided great insight to the hardware/software interface requirements in this project. Additionally, your friendships provided the nucleus of a team that I am proud to have worked with. It is very unfortunate that the LED did not make it into the design!

To Jim Craybill: Your willingness to share your technical expertise in OrCAD and WinCUPL went beyond the call of duty. Additionally, your efforts in producing the PCB are greatly appreciated.

To Professor Alan Ross and Professor Herschel Loomis: Your guidance, patience and tutelage were instrumental in the completion of this portion of the project.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The decline of Department of Defense (DOD) funding of research into radiation hardening and reliability of microelectronics since the end of the Cold War has had serious impact on the price and availability of these parts. This, coupled with reductions in the space budget, has spacecraft designers looking for alternative ways to implement reliable systems with reduced cost, faster design time, and higher performance. One alternative is the use of commercial-off-the-shelf (COTS) devices in place of radiation hardened devices. COTS devices offer spacecraft designers shorter design-to-orbit times, lower system costs, orders of magnitude better performance, and a much better software availability than their radiation hardened (radhard) counterparts. The major drawback to using COTS devices in space is their increased susceptibility to the effects of radiation, both total dose and single event upsets (SEUs).

This thesis will present the implementation of a fault tolerant computer system. The hardware design presented here has two different benefits. First, the system can act as a software testbed, which enables the testing of software fault tolerant techniques in the presence of radiation induced SEUs. This allows the testing of the software algorithms in the environment they were designed to operate in without the expense of being placed in orbit. Additionally, the design can be used as a hybrid fault tolerant computer system. By combining the masking ability of the hardware with supporting software, the system can mask out and reset single processor errors with small cost in time. Both of these concepts will be discussed further in the body of the thesis.

A. THE SPACE ENVIRONMENT

The most important aspect to the designer of an electronic system is its function. Another vital interest to the system designer is the environment the system is going to operate in. If the environment is going to be space, then there are many factors that must be taken into account. The uninformed might think space is a benign environment, but that can not be farther from the truth. The vacuum, temperature extremes, debris, and radiation all interact with the system in one way or another and special efforts must be taken to account for their negative side effects.

1. Vacuum

As you move away from the Earth's surface, the number of particles per unit volume continues to decrease. Starting at roughly 10^{18} particles per cubic centimeter (cm^3) on the surface of the Earth, this number decreases drastically as you increase in altitude. The average particle density from 600 to 1200 miles is only about 100 particles per cm^3. Additionally, as with the particle density, the pressure associated with an altitude decreases as you move away from the surface of the Earth. While the pressure is near 760 millimeters of mercury (mmHg) at the surface of the earth, it is only about 10^{-12} to 10^{-16} mmHg past 1200 miles. These low pressures are sometimes referred to as hard vacuum. [Ref. 1]

One phenomena caused by the hard vacuum of space is called outgassing. Outgassing occurs when a material actually loses mass because molecular gasses trapped inside or on the surface of the material at ground level are pulled out and away by the vacuum of space. A byproduct of outgassing is the cold weld. In the atmosphere, metal surfaces have a thin layer of the molecular gas, which acts as a lubricant when two surfaces are in contact. When the vacuum removes the gas, the metal surfaces are allowed to touch and they will bond together. This issue is one to be dealt with by a materials scientist. The solution to this problem usually relies on the selection of appropriate materials to minimize the effects of the vacuum. [Ref. 1]

2. Meteoroids

Meteoroids and orbital debris pose a risk to satellites. Although there are a large number of meteoroids and space dust near the Earth, most are fairly small and do not pose a large risk of catastrophic damage to a satellite. There have been a couple of satellites that have mysteriously quit working and the conjecture is that the failure might be due to damage caused by impact with meteoroids. With this in mind, the system should be designed to deal with the higher probability of micrometeoroid strikes, where only a small portion of the system is damaged. One option is to design the system using redundancy, a concept that will be explored later in this chapter. [Ref. 1]

3. Temperature

The Earth's atmosphere and the mass of the planet keep the ambient temperatures within a fairly moderate range at the surface of the planet. That can not be said of the space environment. Careful considerations with respect to thermal radiation must be taken into account when designing systems for space application. The skin of a spacecraft exposed to the sun will rise to a very high temperature while the shaded side falls to an extremely low temperature. This causes a very high temperature difference across the space vehicle. With that in mind, microelectronic devices used in systems designed for space must function correctly over a much larger temperature range than required within the Earth's atmosphere. [Ref. 1]

4. Radiation

Radiation is the emission or propagation of waves or particles. It is the key element of the space environment that our design is taking into consideration. High-energy charged particles can cause damage or disruptions, which are discussed later in the chapter, in microelectronic devices. These particles are either ions or photons. Ions, except for the neutron, have both a charge and mass associated with them. There are basically four types of ions: electrons, neutrons, light, and heavy. Light ions have a very low mass, such as protons, which are hydrogen atoms with the electron missing, and alpha particles, which are helium atoms with both the electrons stripped off. Heavy ions are any element heavier than helium with one or more electrons missing. Unlike ions, photons have neither mass nor charge. They are just very short wavelengths of light, such as X-rays and gamma rays. [Ref. 2]

There are several contributors to the radiation effects near the Earth. The largest contributor to a device's total dose is from particles trapped in the Earth's geomagnetic field. These trapped particles make up an area known as the Van Allen Belts. Any satellite in orbit is subject to effects from the Van Allen Belts. Another contributor is solar particles. Due to the high temperatures of the sun, many particles have enough energy to escape the sun's gravity. Those particles continuously flow across the Earth in what is called the Solar Wind. The sun continually moves through cycles of solar activity, where a single cycle takes approximately eleven years to complete. Another source of radiation is galactic cosmic rays. These are heavy ions produced by events, such as exploding stars, outside our solar system. [Ref. 2]

When radiation interacts with microelectronic devices, it is either absorbed into or passes through the semiconductor leaving a path of ionization. The radiation has four major different types of effects on the semiconductor: Total Dose Effects, Dose Rate Effects, Displacement Damage, and Single Event Effects (SEEs). [Ref. 2]

Total Dose Effects are device failures caused by the lifetime sum of radiation absorbed by the device. Similarly, Dose Rate Effects are where the device fails to function at a particular radiation rate, or number of particles per unit of time. Displacement Damage deals with nuclear interactions between the particle and the semiconductor. The radiation changes the nuclear makeup of crystal atoms within the semiconductor, which leads to device failure. [Ref. 2] Finally, SEEs are the effects of charged particles changing the state of transistors in the circuit. SEEs are the radiation effect we are most interested in and are discussed in greater detail in the following section.

B. SINGLE EVENT EFFECTS (SEE)

The main difference between SEEs and the radiation effects discussed in the last section is that SEEs are not necessarily destructive in nature. SEEs are caused when an ionized particle deposits enough charge to cause transistors to change state in a device. In most cases, the transistor only changes state long enough for the charge to be absorbed back into the system and then resumes its original state. The transistor's state change can lead to latchup in parasitic transistors, be purely transient, or be latched into a storage element. These three main types of SEE in Complimentary Metal Oxide Semiconductors (CMOS), which is currently the most popular type of circuit implementation for microelectronic devices, are discussed in the following sections. [Ref. 2]

1. Single Event Latchup (SEL)

When CMOS field effect transistors are fabricated near each other on a single chip, one of the unwanted byproducts is a pair of vertical bipolar junction transistors. An SEL is caused when a charged particle passes close enough to this circuit to bias the two parasitic transistors on. This creates a very low impedance path to ground, which has two possible outcomes. If the current drawn through the parasitic transistors generates more heat than the device can dissipate, it will be destroyed. Even if the device can dissipate the heat, the large amount of current drawn through the parasitic transistors prevents the remainder of the circuit from operating correctly, which is a non-destructive SEL. The normal manifestation of a non-destructive SEL is that of a hung system, which requires a system reset for recovery. [Ref. 3]

2. Single Event Transient (SET)

Single Event Transients are unexpected, short duration changes in the output value of a combinational logic circuit due to the influence of a charged particle. SETs are not detrimental to the system. They only adversely effect the system if the pulse change is near the clocking edge and is long enough to meet the setup and hold times of the next storage unit in the cascade of stages. If the SET meets these criteria, then it manifests itself as a single event upset (SEU). [Ref. 4]

3. Single Event Upsets (SEU)

An SEU is any unwanted value change in a memory cell, whether it be a latch, register, or cache cell, that is caused by charge introduced into the circuit by radiation. In microprocessors, SEUs are usually categorized into one of two error types: program flow errors and data errors. Program flow errors are errors that occur in the program counter (PC), control logic, or any other register that deals with the state of the processor. Data errors are usually confined to the registers and data cache. These two types of errors are not necessarily exclusive. A data error could occur in a register that is later used as a jump address. When the PC jumps to the address held in that register it is in the wrong location and begins to execute incorrect code. [Ref. 4]

C. COMMERCIAL-OFF-THE-SHELF VS. RADIATION HARDENED DEVICES

The radiation effects discussed in the previous sections, with the exception of SETs and SEUs, are destructive in nature. The main way of reducing their effects is by using radiation hardened (radhard) devices or providing shielding. A radhard device is one that is specifically designed to be able to withstand higher amounts of radiation than standard commercial parts.

In a direct comparison of commercial-off-the-shelf (COTS) parts to radhard parts, the first question someone is bound to ask is 'why would we use non-hardened parts in such a harsh environment?' This section gives some insight into the answer of that question.

1. Cutting Edge Technology

As alluded to in the introduction, companies developing and marketing radhard devices are on the decline. Without DOD budget support for research into this area, there is not sufficient demand from commercial sector customers to motivate companies to sink large sums of money into research on these devices. For those reasons, the technology of radhard devices is lagging behind state of the art technology by up to two or more generations. As an example, a state of the art processor, the 600 MHz Intel Pentium III, AMD K-6 II, or a RISC design, depending on your preferences, is available off the commercial shelf. The nearest competitor on the radiation hardened shelf would be a 66 MHz 486 processor. This is a whole order of magnitude difference in processor capability. [Ref. 5]

2. Faster Design-to-Orbit Time

Because many manufacturers of radhard devices offer them as a secondary market, many do not readily stock the parts. This causes a delay in the design and test phase of the project. With a move to COTS devices, the order delay is completely erased. Most of these devices are available from multiple vendors, which gives the designer even more leverage in selecting a vendor. Additionally, data on radiation testing for many devices is now becoming available to help the designer make an informed decision on the correct part to use based on the environment it is to be placed in. [Ref. 5]

3. Reduced Cost

Profit or lack thereof is one of the main reasons companies have moved away from the manufacture of radiation hardened devices. The low demand for these devices keeps the price hundreds of times higher than the commercial models for several different reasons. Since radiation hardened devices employ different techniques in their design to reduce the susceptibility to effects from ionized particles, they tend to be larger than the non-hardened devices. This lowers the number that will fit on the wafer in the first place and increases the probability that the devices will have defects. The reduction in yield can be attributed to two main factors. The first is the higher probability of defects in each device mentioned before and the second is from the limited number of runs of the fabrication process. Because yield normally increases with the number of production runs and radhard devices account for a small number of fabrication runs, radhard devices normally do not show large improvements in yield. Since the demand for commercial devices is much higher, the manufacturer can adjust the fabrication process over a large number of runs to increase the yield, which results in lower cost to the consumer. [Ref. 2]

D. PURPOSE

The goal of this research is the implementation of a fault tolerant computer system using COTS microprocessors that is able to accurately compute in the presence of radiation induced SEUs. This research specifically concentrates on the ability of the system to detect and mark or correct SEUs in a microprocessor in real time. This work does not address error detection and correction (EDAC) of memory systems, which has been previously researched [Ref. 6] and is a topic left for future inclusion.

There are two major benefits from this study. First, the system can act as a fault tolerant software testbed. The processor is monitored for an SEU. When one is detected, a time stamp is generated and the Operating System is observed in its handling of the error. This allows the testing of the software algorithms in the environment they were designed to operate in without the expense of being placed in orbit.

The system can also be used as a hybrid fault tolerant computer system. In this use, the processor is also monitored for an SEU. When one is detected, the faulty data is corrected in the processor and it continues to execute its instructions from the corrected data. A major advance in this implementation is the ease with which the error can be corrected. The normal mode from previous research in these types of systems has been to just reset the processor when an SEU was detected, effectively loosing all computations done up to the time of the SEU.

E. THESIS ORGANIZATION

The organization of this thesis follows closely to the design approach used in developing the system. Chapter I has been a brief introduction into the environment that the system will be operating in. Chapter II is background material on research that set the foundation for this design. Chapter III presents the hardware design of the system and points out changes from the previous design. Chapter IV contains the design of the programmable elements of the system, which include the Voter Modules and the System and Memory Controllers. Chapter V presents the steps that were taken to have the printed circuit board fabricated and the design review that was conducted after the board was manufactured. Chapter VI presents the conclusions developed during this research and discusses topics for follow-on work.

II. BACKGROUND

The study of fault tolerant computing systems has been going on for many years. In fact, many early analog computer systems were designed with duplicate processing units because of their propensity to have errors. The EDVAC computer, which was designed in 1949, had duplicate Arithmetic Logic Units (ALUs) that continued to compute as long as their results agreed. As computer systems moved further into the digital era and became more reliable, fault tolerant designs took a back seat to designs that improved performance and speed. When computers started to perform critical tasks in systems, designers had to once again focus on fault tolerance. [Ref. 7]

The purpose of this chapter is to provide the reader a brief background for this project and the key issues that make it important. The chapter starts by describing the general concept of redundancy and focuses in on the design from which this system was implemented.

A. REDUNDANCY

The fundamental concept for implementing fault tolerance is redundancy. In its most basic form, redundancy is the addition of resources beyond those required by the system for normal operation to achieve reliability, availability, or safety. There are four different kinds of redundancy that can be implemented in a system. They are software, information, time, and hardware redundancy, each of which is discussed below. [Ref. 7]

1. Software Redundancy

One of the key attributes of software redundancy is that it requires a minimal amount of additional hardware to support it. Three typical software redundancy techniques are consistency checks, capability checks, and n-version programming. Consistency checks use a-priori knowledge of the data set to check it to ensure that it is consistent with the expected values. Capability checks will test functional units of the system to ensure they are in working order. While both of these methods are software tests of hardware, n-version programming is a software test of software. Different programmers code different versions of the same function. All of the versions are executed and their results compared to ensure that all the results are the same. [Ref. 7]

2. Information Redundancy

Information redundancy provides additional bits along with the data to allow for checking that the correct values are received at the destination. The additional information can be small enough to allow error detection or robust enough to allow multiple bit corrections. Two good examples of information redundancy are the parity bit and the Hamming Code. The parity bit is a single bit appended to the data set that allows for error detection. Parity can either be even or odd. If even parity is selected, the number of ones in the data set and the parity bit should add up to an even number. If odd parity is selected, the ones should add up to an odd number. If they do not, there is an error in the information. The limit of the parity bit is its inability to determine which bit is in error. The Hamming Code appends multiple bits to the data set, which allows for error detection and correction by pointing out the faulty bit. The additional bits appended by the Hamming Code are actually weighted parity bits. By determining which Hamming Code bits are in error, the faulty data bit can be determined and corrected. [Ref. 7]

3. Time Redundancy

One of the costs associated with all forms of redundancy is the large overhead. This overhead can be paid in the form of additional hardware, weight, and power consumption. In some cases, these assets are not available for trade. When that happens, another form of fault tolerance must be found. In systems where speed is not a critical issue, time redundancy can be used. The premise behind time redundancy is that the same calculations can be performed serially multiple times and the outputs compared to check for errors. This requires the system to save the state before the calculation, perform the calculation and save it, make a context switch back to the beginning of the calculation, perform it again, and then compare the results of the two different calculations. [Ref. 7]

4. Hardware Redundancy

When you mention redundancy to people, hardware redundancy is typically the first thing that enters their mind. This form of redundancy is regaining popularity as the cost for replicating hardware is getting less and less expensive. There are basically three forms of hardware redundancy: passive, dynamic, and hybrid. Passive redundancy uses fault masking to keep the fault from propagating out of the current process. This is implemented using multiple modules and voting hardware. It does not require any actions from the system or operator. Dynamic redundancy monitors the outputs of the modules looking for faulty units. When one is detected, the system removes the faulty unit from the system and replaces it with a good one if it is available. The hybrid approach uses portions of both the passive and dynamic approaches. [Ref. 7]

A subset of passive hardware redundancy is the Triple Modular Redundant (TMR) system. As the name implies, the TMR system takes the outputs of three replicated modules and compares them in a voting unit. The voting unit passes the most common input to the output, essentially masking out any single fault. The heart of a TMR system, the voting unit, and its truth table are shown below in Figure 2.1 and Table 2.1.

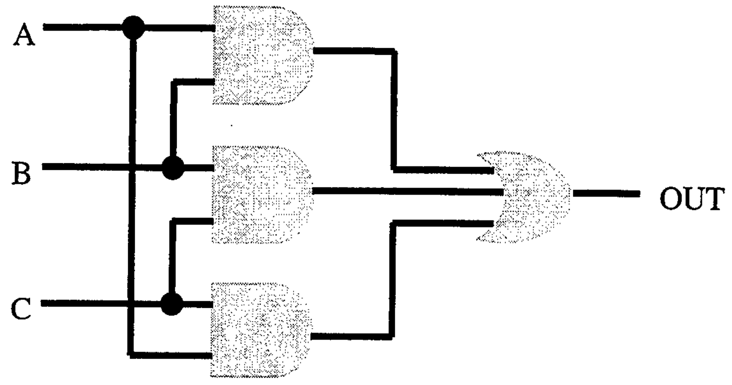


Figure 2.1. 3-Bit Majority Voter Logic Diagram. After Ref. [7]

A	0	0	0	0	1	1	1	1
B	0	0	1	1	0	0	1	1
C	0	1	0	1	0	1	0	1
OUT	0	0	0	1	0	1	1	1

Table 2.1. 3-Bit Majority Voter Truth Table

The table shows that anytime two or more inputs have the same logic value, either a zero or one, then that value is propagated to the output. This system can also be extended to an N input model. As N gets larger, the logic required to realize the circuit and the added levels of delay get prohibitively large. Typically N is held to three or five.

5. TMR Implementations

As stated before, by placing the three modules in parallel and voting their outputs, a TMR circuit can be implemented. The basic TMR circuit is shown below in Figure 2.2. The inputs and output of the modules do not have to be single values. There can be X inputs and Y outputs associated with each module. All that needs to be done is place Y single bit voters in parallel connected such that they vote each of the respective Y outputs.

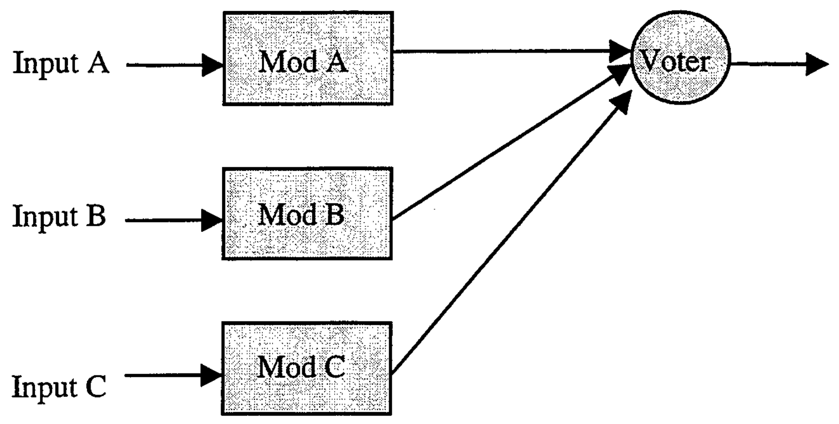


Figure 2.2. Basic TMR Circuit Implementation. After Ref. [7]

The single voter on the output of the TMR system is a single point of failure. That is, if the voter fails and generates or propagates an error, then the error will be propagated throughout the circuit. This could become a major problem in cascaded circuits. To solve this problem, the voters at the end of each stage can also be triplicated. [Ref. 7] An example of this is shown in Figure 2.3.

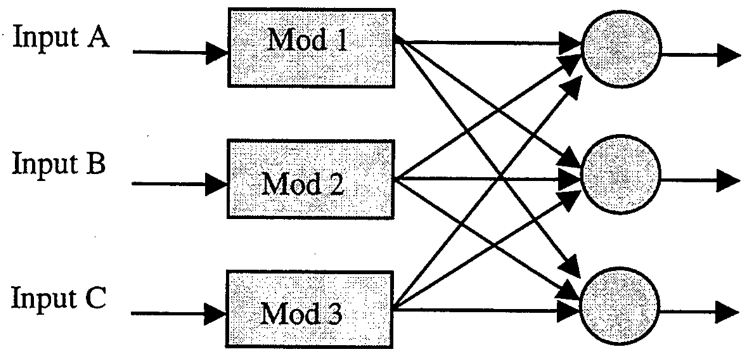


Figure 2.3. Triplicated TMR Voters. After Ref. [7]

B. SOFTWARE VS HARDWARE REDUNDANCY

The technology associated with very large-scale integrated circuit (VLSI) design has advanced at a phenomenal pace. Systems that used to be produced as multiple chip designs are now compacted into systems on a chip. Much of the current hardware design emphasis is placed on reducing the transistor size in order to increase the capability and decrease communication times within the chip. Since the communication times between chips in a system has not progressed at the same rate as internal communication, the addition of logic layers between the processor and its peripherals would require the system bus clock to operate at a reduced frequency. Because of this, general trends have moved the reliability issue to software.

When using software fault tolerant techniques, as in Checkpointing, the software must run the same segment of code a minimum of two times. If the results of the first two runs match, the processor can continue where it left off after it has saved its internal state. If the results do not match, the processor must run the code segment an additional time to try to determine which of the first two runs were correct and then save the correct internal state. This occupies over half of the processor's execution cycles in context switching and rerunning code. [Ref. 8]

The option being investigated in this research is a move back to hardware redundancy. By applying the TMR technique discussed earlier to the microprocessor, the outputs of the processors can be combined through a voter. If an error occurs in one of the processors the voter will mask it out and the bus cycle will complete normally. The two good processors will then reset the errant processor to their good state and then all three processors will continue executing the program.

A direct comparison of the two options is very difficult. Although the TMR design looks like it can save over half the time required to perform a function, it can not operate at the same clock speed that the Checkpoint model can. The TMR system has to place additional logic between the processor and the memory space to perform the voting operation, which decreases the maximum clock speed at which the processor can operate.

C. TMR MICROPROCESSOR DESIGN

The framework for the system implemented in this work was originally designed and simulated using Verilog by Lieutenant John C. Payne, Jr., USN, as a Fault Tolerant Computing Testbed. The remainder of this section is a brief synopsis of his TMR Testbed Design. For a more detailed explanation, please see Ref. [9].

1. Processor Selection

Lt. Payne began his design with the logical step of selecting a microprocessor. In his selection process he took into account such factors a COTS vs. Radhard, CISC vs. RISC, size, pinout, power, bus width, memory size, speed, and multiple chip vs. single chip implementations. His research led him to select the MIPS R3081 RISController produced by Integrated Device Technologies (IDT). This device was selected over AMD's AM29000 and AM29050; IBM and Motorola's PowerPC 603e, 604e, and 750; and IDT's R36100, R4650, and R5000. A detailed description of the R3081 can be found in Ref. [10].

2. Hardware Design

With the processor selected, the next step was to integrate all the peripheral components required for the R3081 to function as a TMR computer system. For comparison purposes, a block diagram of a single processor system and a TMR system are provided on the following page in Figure 2.4 and Figure 2.5, respectively.

From comparison of Figures 2.4 and 2.5, the architecture of the TMR implementation has relatively few changes from the single processor design. The major additions are CPU B and CPU C along with the Address, Data, and Control Voters.

The processors are connected in such a way that the Operating System acts as if there is only one processor in the system. The processors are kept in lock step from boot up by executing the same instructions in parallel. The processors Address, Data, and Control busses are then routed to their respective voters. The voters perform a majority vote on the signals and pass them on to the Memory Space and Memory Controller as in a single processor system. If an error is detected in a voter, the Memory Controller generates an interrupt. [Ref. 9]

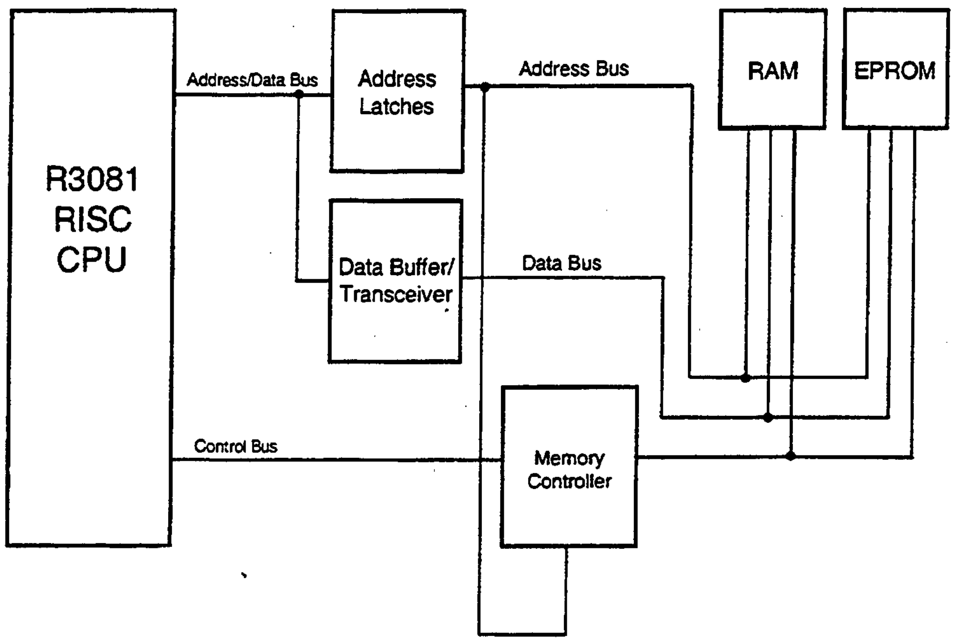


Figure 2.4. Simple R3081 Board Design. After Ref. [11]

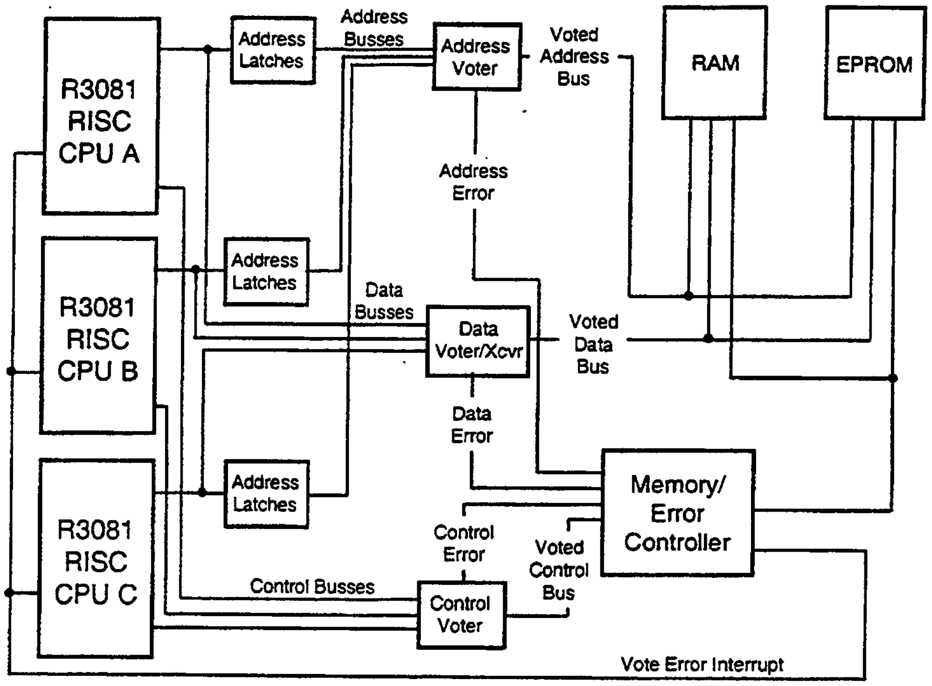


Figure 2.5. TMR R3081 Board Design. From Ref. [9]

3. Software Design

The voters allow the fault generated in one of the processors to be masked out and the bus cycle to complete correctly, but that does not completely remedy the situation. The processor that had the error is now out of synchronization with the other two processors. That is where the software design of the Interrupt Handler comes into play.

When a voter detects a miss compare on its inputs, it signals the Memory Controller, which asserts an interrupt input to the processors. The Interrupt Handler resets the invalid processor in a very simplistic way. It starts by saving the processor's internal registers to memory. Since all three processors will be executing the instructions, the Data Voter will mask out the invalid data from the corrupted CPU. The Interrupt Handler then reloads the processors internal registers from memory. This puts the corrupted processor back into synchronization with the other two processors. The Interrupt Handler then acknowledges the interrupt and returns from the exception. The processors then continue execution with the next instruction.

In order to determine which processor was corrupted, the internal registers of each processor must be examined. The data must be captured prior to being voted or the error is lost. By placing First-In-First-Out (FIFO) Registers on the address, control, and data busses between the processors and voter, each processors internal state can be captured in its corresponding FIFO. The arrangement is shown below in Figure 2.6.

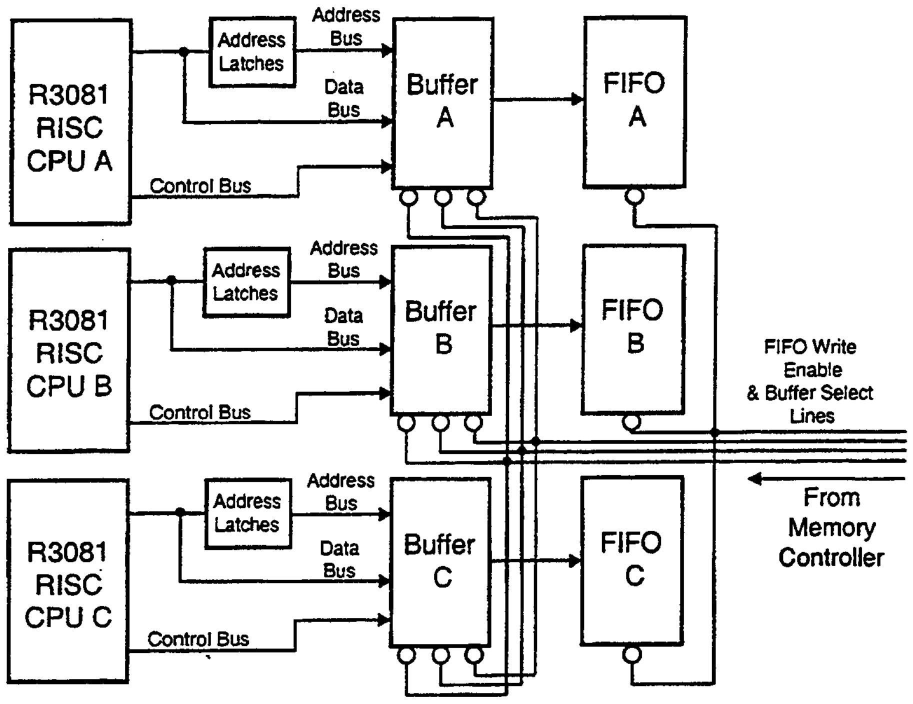


Figure 2.6. TMR FIFO Interface. From Ref. [9]

Lt. Payne's design and simulations prove the fundamental concept of a working TMR R3081 system. The next logical step is to implement the system in hardware. The next two chapters examine the transition of this design from simulation to hardware. Chapter III presents the hardware implementation, while Chapter IV presents the programmable logic synthesis. Because of hardware limitations and additional system requirements, the transition is not a one-to-one process.

THIS PAGE INTENTIONALLY LEFT BLANK

III. HARDWARE DESIGN SPACE

The process of designing a complicated electronic system is very seldom linear in nature. That is, the system originally conceived is seldom designed, simulated, implemented, and manufactured without changes. Many factors affect the process as the design matures, which can cause changes in areas of the design that were once thought to be complete. This results in feedback loops in the design process. This design has not been an exception to these effects. It has gone through several revisions in the implementation and manufacturing phases due to feedback from several different factors, including changes in requirements and parts availability. This chapter presents the final hardware implementation and explains design changes where major deviations from the simulation model were required.

A. DESIGN PARAMETERS

As discussed in Chapter I, the most important aspect of the system the designer must be familiar with is its function. Chapter II presented a simulated design by Lt. John Payne, Jr. [Ref. 9] that fulfilled all the functional requirements, but focused primarily on validating the concept of TMR using the IDT R3081 microprocessor. The next step in the design process is the hardware implementation. This implementation was developed by making modifications to Lt. Payne's design and using three general functional applications: use as an Evaluation Board, easy adaptation to a space flight board, and the ability to be tested in a particle accelerator.

1. Evaluation Board

The evaluation board design parameter adds value by introducing flexibility and visibility to the design. For flexibility, socketed parts and programmable logic are used. By using socketed parts instead of parts that must be soldered to the printed circuit board (PCB), the designer enables the user to quickly and easily exchange integrated circuit (IC) chips. This allows easier replacement of damaged parts and requires fewer vias when routing traces on the PCB layout, when compared to surface mount devices. Upgrades to the system can also be easily incorporated if the new IC chip is pin compatible with the old one. The most common use of this advantage would be upgrading to faster parts.

Using programmable logic to interface the processors and the peripherals is another tool the system designer can use to increase flexibility. Unless the same manufacturer builds the processor and peripheral, there is normally a requirement for additional logic to be placed between the two components to allow them to function together properly. The additional logic is usually referred to as "glue logic." By using a programmable device to perform the glue-logic function, the system can easily be expanded to incorporate system updates and changes.

Since evaluation boards are normally used to investigate new concepts, it is important to be able to capture data from as many internal signal lines as possible. Today's PCBs are normally fabricated using multi-layer designs, making it difficult to capture signals residing on internal routes. Connecting the essential signals to a test connector, which makes them visible to the user, can alleviate this problem.

2. Space Flight Board

Although it may seem that designing the initial implementation of a system with such a futuristic goal would increase the complexity of the design, in this case it actually narrowed the scope on many of the part choices. The premise of this design parameter was to focus on parts, other than the microprocessor, that were available in both commercial and rad-hard versions or parts that showed a reduced susceptibility to SEUs. The pin compatibility and exact functionality between the commercial and rad-hard versions of devices reduces the process of upgrading to a flight ready board to a simple IC exchange with minor modifications to the timing parameters of the glue logic. As for parts showing reduced susceptibilities to SEUs, different technologies used to implement devices can have drastic differences in their SEU susceptibility, such as in the case of static or dynamic RAM. When these choices were available, the technology offering the lower susceptibility was used.

3. Accelerator Testing

One of the key functions of this system is its ability to test fault-tolerant software. In order to achieve that function, the user must be able to operate the system in a particle beam, which places added requirements on the implementation of the system. First, the system must be able to communicate with the host interface over distances of fifty feet, due to the shielding and physical layout of the testing facilities. Because heavy ion testing requires the system to be placed in a vacuum chamber, the device must be able to physically fit within the chamber and be reset remotely. The remote reset allows the user to recover a "hung" system without having to bring the chamber back to local pressure, reset the system, and then reapply the vacuum, which wastes expensive chamber/facility time.

B. SYSTEM OVERVIEW

The final block diagram for the TMR R3081 system as implemented is shown in Figure 3-1. Compared to the simulated system depicted in Figures 2-5 and 2-6, there are several differences. This section will discuss those differences.

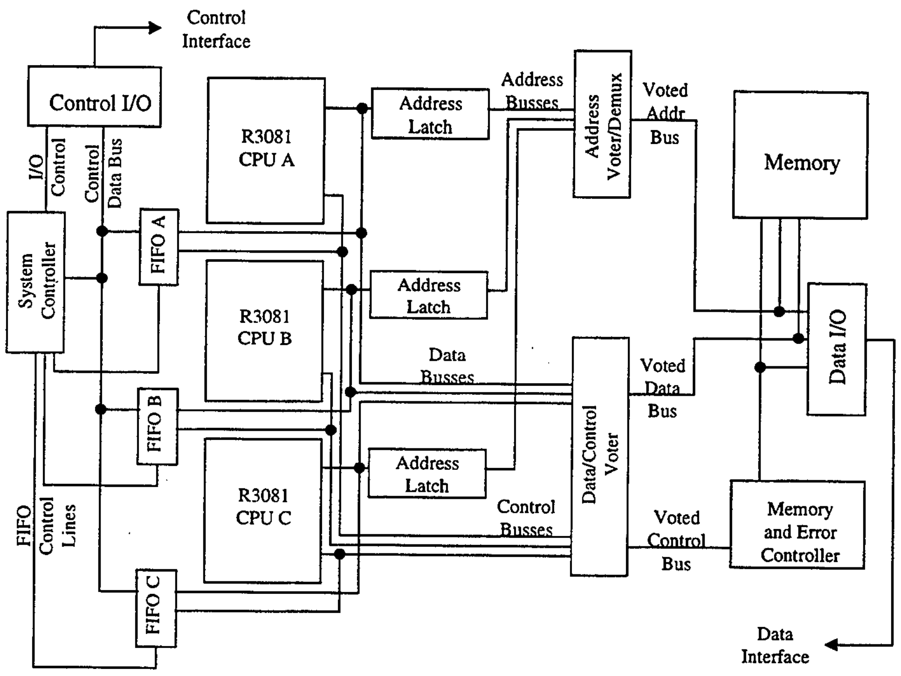


Figure 3.1. TMR R3081 Block Diagram

The most important difference between the two designs is the addition of a System Controller to the implementation, since the Verilog® design suite acted as the

System Controller during simulation. For the implementation, an FPGA was selected to perform the System Controller function. Its design is explained in Chapter IV.

The other main design change is concerned with the I/O interface. Since the scope of the design in Ref. [9] did not require the loading and running of user programs, an interface to provide that functionality was not included. Additionally, the simulation used a System Interface module to read out the collected FIFO data. Although the System Interface was expected to be a laptop or similar system, its interface was simulated using a 104 bit wide interface, which could not be directly implemented into hardware. This left a wide latitude in the implementation of the I/O interface. For reasons discussed below, a dual-port interface was selected.

Although this thesis focuses on hardware implementation, two other major efforts on this project were going on in parallel. Captain Kim Whitehouse, USMC, and Lieutenant Susan Greoning, USN, conducted research into the selection of an operating system for the TMR R3081 and the design of a Human Computer Interface (HCI). Their research garnered the selection of VxWorks as a real-time operating system and the design of a user interface for a software testbed which is presented in Ref. [12]. The important aspect of the close association of the hardware and software designers is the ability to effectively integrate the two designs, which is why several hardware design decisions are based on the software interface requirements.

The functionality of the HCI plays a significant role in determining the design of the I/O interface. The HCI controls the TMR's mode of operation, provides for a remote reset, enables the user to download and run programs, and also provides a path for data collected in the FIFOs to be returned for analysis. Serious consideration was given to the use of a single communications port for this interface. Difficulties arose in the design when trying to implement the remote reset function with a single port. Because the CPUs control the transfer of data in a single port device, a "hung" system would not be able to reset because it could not respond to the reset signal at the port. In order to achieve this function a second port, controlled by the System Controller, was added to the implementation.

With the decision made to use two ports, the next step was to determine what functionality was associated with each port. To help distinguish the two ports, which were depicted back in Figure 3.1, the processor-controlled port was named the Data Port and the System Controller controlled port was named the Control Port. Since the ability to download and run programs does not require interface with the System Controller, it was left on the Data Port. In order to free the processors from the overhead associated with transmitting the large FIFO data blocks and to provide the System Controller with direct access to control and resets, these functions were assigned to the Control Port. This separation lets the user interact with the downloaded program through the Data Port and interact with the TMR hardware through the System Controller on the Control Port.

Now that the major changes in the implementation from the simulation have been presented, the rest of the chapter discusses the individual pieces of the design. Section C presents the design and implementation of the system support elements that were not required during the simulation phase. The microprocessor and address latch selection is then addressed in Section D. The design of the Memory Space and its components follows in Section E. Finally, Section F presents the design of the FIFO buffer interface.

C. SYSTEM SUPPORT ELEMENTS

The detailed analysis of the TMR implementation starts with the system support elements. These portions of the system are critical to the systems functionality, setup, and testing, but are seldom included in simulations. One of the benefits of simulation is the ability to only include the required portions of the system. The implementation must account for all the elements that must be added to support the main system that were not included in the simulation, such as pull-up and pull-down resistors, decoupling capacitors, clocking and reset systems, mode select, and test connectors.

1. Discrete Components

One important change from a simulation environment to a hardware implementation is the inclusion of the discrete resistors and capacitors used in the circuits. When control signal lines are driven by devices that can tri-state their outputs pull-up or pull-down resistors are required to maintain the signal line at a known value. This prevents the device from misinterpreting the driver going into the high impedance state as an active signal. Additionally, decoupling capacitors are required to reduce the power fluctuations seen on the voltage bus during switching and to keep AC noise off the DC bus.

2. Timing Interface

The Timing Interface consists of two different clocking systems, one for the CPUs and one for the two universal asynchronous receiver transmitters (UARTs), which are used to transmit data through the Data and Control Ports. The schematic for the Timing Interface is given in Figure 3.2. The two oscillators and the buffer/driver are all socketed to allow the user to easily change the oscillator frequency or the buffer/driver chip. The frequency selections and buffer/driver logic selections are discussed below.

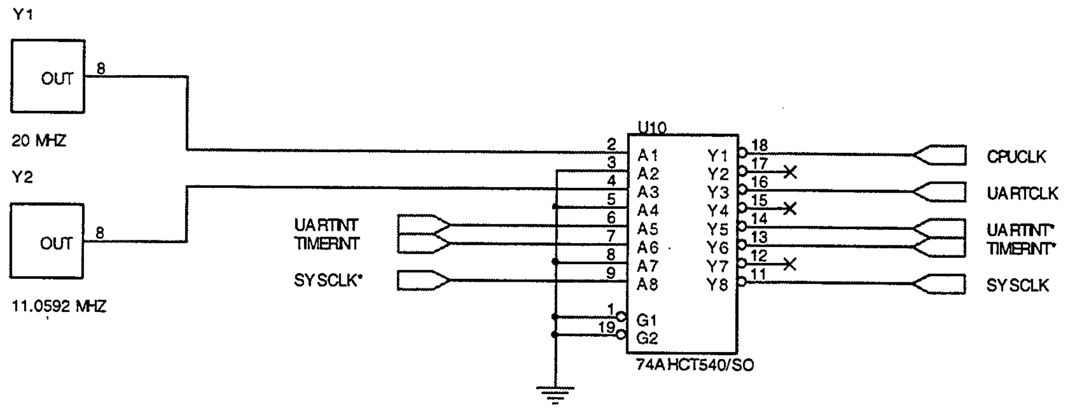


Figure 3.2. Timing Interface

a) CPU Oscillator

For reasons discussed later, the 50 MHz model of the microprocessor was selected for this implementation. Ref. [13] requires the ClkIn signal, or CPU clock, to have a period between 40 ns and 50 ns, which translates to between 20 MHz and 25 MHz. This is because the 50 MHz R3081 is restricted to the 1x Clock Mode, where the ClkIn signal is passed to an internal clock doubler and converted to a double frequency clock for signals internal to the processor. The CPU oscillator selected for this implementation was the 20 MHz model, which provided a slightly longer clock period to account for the added propagation delays of the Voter FPGAs.

b) UART Oscillator

The UART Oscillator provides a reference input to the two UARTs that is used to generate the clock signals that control the serial communications between two terminals. For the two devices to communicate, their UARTs must be set up to transfer and receive data at the same rates. In order to accomplish this, the UARTs provide an internal register that can be loaded with a divisor to generate any baud rate equal to or lower than the reference signal. The divisor is usually used to generate one of the industry standard baud rates, such as 38.4, 57.6, or 115.2 kbps. This allows the user flexibility when selecting the interface frequency.

The design goal was to provide the highest data transfer rate possible within the constraints of the I/O interface. Higher data transfer rates reduce the transfer time required for user programs and CPU FIFO contents. The original oscillator frequency selected was 16 MHz, which is the maximum allowed oscillator input to the UART. This frequency allows the UART to transmit at its maximum rate of 1Mbps. Unfortunately, other system constraints discussed in the I/O portion of the Memory Space section of this chapter could not support the maximum transfer rate.

Since the maximum baud rate could not be supported, an attempt was made to maximize the number of standard baud rates the system could support. This led to the use of an 11.0592 MHz oscillator. This frequency allows for standard baud rate generation up to 115.2 kbaud, which is the maximum transfer rate currently supported by the HCI serial port.

c) *Buffer/Driver Logic*

The main purpose of the Buffer/Driver chip is to buffer the low power oscillator signals into output signals strong enough to drive the CPU and UART clock inputs. The CPUCLK signal that emerges from the buffer/driver is the timing reference signal used by the CPUs. Since the input signal to the buffer from the oscillator is not referenced by any other component, the phase and delay of the buffered signal compared to the oscillator signal does not matter. The same applies to the UARTCLK, since its frequency only needs to be matched to the device it is communicating with.

Most buffer/drivers come in packages of eight. In order to minimize the chip count, other uses of vacant glue logic chips were considered. In this case, three additional buffers could be used if the buffer/driver inverted its inputs. Two interrupt signals, UARTINT and TIMERINT, are generated by their respective devices in positive logic and the buffer/driver inverts them to the negative logic required by the CPUs. Additionally, the SYSCLK* signal generated by the CPUs as a reference signal for bus interface transactions is inverted by the buffer/driver to get the SYSCLK signal used by the memory controller state machines out of phase from the CPU reference clock. This allows the memory controller to meet the setup and hold times required by the CPU for read and write cycles without adding additional wait states beyond those required by memory access times.

There is a very large pool of inverting buffer/drivers to choose from. The design goal used in the selection of the buffer/driver was a CMOS implementation with less than 12 ns propagation delay. The CMOS implementation reduces the static power dissipated by the device, which is also one of the major considerations when selecting parts for a space flight board. The restriction on the propagation delay was needed to ensure the Read Clock Enable and Acknowledge signals reached the CPUs soon enough to meet the required setup times. The Texas Instruments (TI) 74AHCT540 was selected because it met these requirements [Ref. 14]. Chapter IV will present a detailed analysis of these timing signals.

3. Reset System

Most computer systems have a single level reset circuit that is activated when power is turned on or by a push button switch. The TMR implementation uses a two level reset system that can be activated three different ways: power on, remotely, or by push button switch. The Reset System design evolved from the system requirement of being remotely reset and the use of serially programmed FPGAs. The main reason a two level reset was implemented was to allow the system to be reset without having the delay associated with reprogramming the FPGAs. A schematic diagram of the Reset System is shown in Figure 3.3.

When power is initially applied to the system, the Supply-Voltage Supervisors (SVSs) sense the level of the supply voltage. When the supply voltage crosses a point near 3.6 V, their RESET outputs go active and the timer function is initiated, as long as the RESIN* input is not active. The timer function is determined by $t_d = 1.3 \times 10^4 * C_t$, where t_d is the delay time and C_t is the total capacitance in the timer circuit. [Ref. 15]

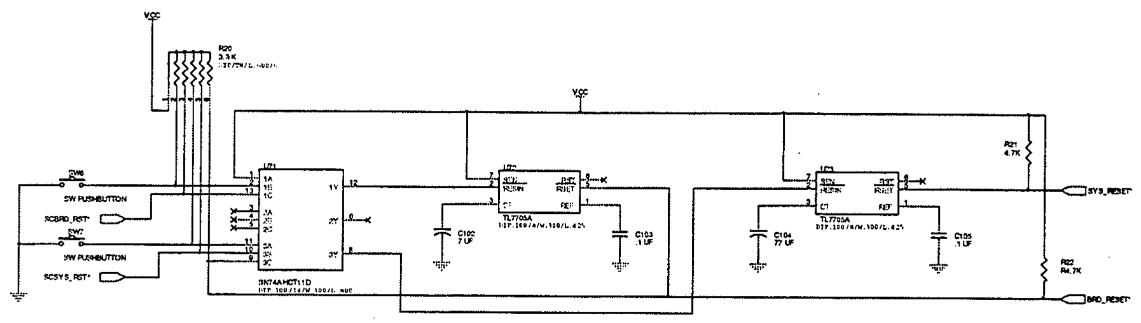


Figure 3.3. TMR Reset System

The first SVS, U72, uses a 7 μF capacitor in its timer circuit to hold the BRD_RESET* output low for 91 ms. The reset pulse allows the FPGAs to recognize the reset signal and setup for their initialization. The BRD_RESET* signal is also routed back to one of the inputs of a 3-input AND gate. The AND gate combines the three input signals to the second SVS, U73. This feedback prevents the timer of the second SVS from starting until the first timer has finished. The FPGAs use the time delay provided by the second SVS, 1 second using a 77 μF capacitor, to load the serial configuration data from their PROMs and to initialize. The one-second delay for loading clearly meets the 200 μs pulse required by the processor [Ref. 10]. If the two step approach were not used, the processors would attempt to make memory accesses through the FPGA vote logic, but no signals would pass through the FPGA because all the inputs and outputs of the FPGA are in a high impedance state during initialization.

After the power on reset (POR) is completed and the board is operating, there are two levels of reset the user can initiate. The first is a Board Level Reset. This reset goes through the complete POR routine. The second level of reset is a System Level Reset. It does not reset the FPGAs, but resets all the remaining devices in the system.

Each level of reset has two methods of activation. The first method is through a normally open push button switch. Since the switch is normally open, the input to the AND gate used by the switch must be pulled-up to Vcc. When either switch is closed, an active low signal is applied to the respective AND gate, which causes the appropriate RESIN* input to go active. The second method is initiated remotely by the HCI. The HCI can send a new control word to the Control Register in the System Controller, which is described in Chapter IV. The System Controller drives the SCBRD_RST* and SCSYS_RST* inputs to the reset logic. When the System Controller activates one of these inputs, the appropriate level reset is initiated remotely.

4. Mode Select

The IDT R3081 processors have several features that are selected during system reset. The signals used to select these features are latched into the processor off the Interrupt bus by the rising edge of RESET*. The input signals and the functions they select are given on the following page in Table 3.1. [Ref. 10]

Interrupt Pin	Mode Feature
INT(5)*	CoherentDMAEn*
INT(4)*	1xClockEn*
INT(3)*	Half-frequency Bus*
SINT(2)*	DblockRefill
SINT(1)*	Tri-State*
SINT(0)*	BigEndian

Table 3.1. Mode Selectable Features. From Ref. [10]

Since the Interrupt bus must provide both the Reset Vector and the Interrupt signals, a system to multiplex the two signals had to be designed. A tri-statable non-inverting buffer/driver was used to select between the two signals. The Interrupt bus is routed through the buffer. During reset the buffer is held in the tri-state mode, effectively blocking the interrupt signals. This allows the signals selected by pull-up and pull-down resistors to be latched into the processor. After the reset is complete, the System Controller activates the INTEN* signal, which allows the interrupt signals to drive the processor interrupt inputs. A schematic of this system is shown in Figure 3.4.

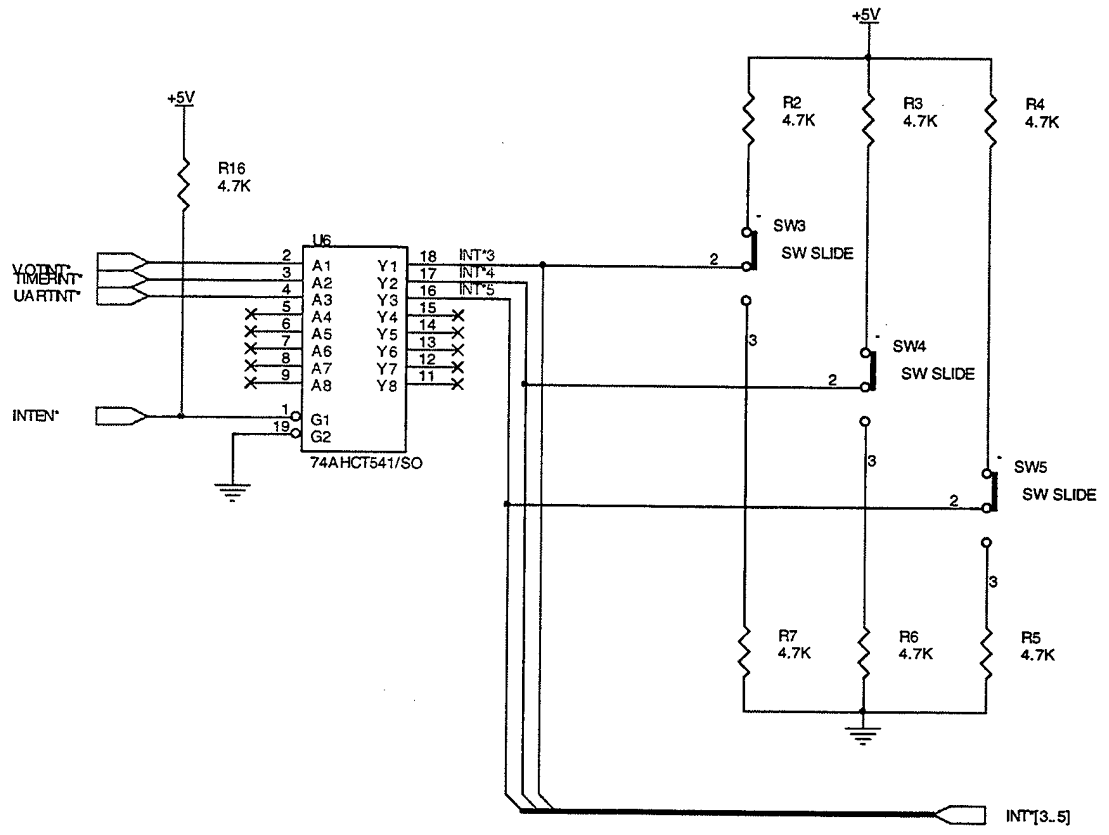


Figure 3.4. Interrupt/Mode Logic

The three synchronous interrupt inputs, SINT*[0..2], are hardwired to Vcc, since there was no need for the user to change them. They are hardwired to select Big Endianess, not Tri-stated, and Data Block Refill. Since VxWorks requires Big Endian functionality [Ref. 12], this mode selection was hardwired. Since there is no need to Tri-

State the processors outputs until another reset, essentially disabling the processor, this input was tied inactive. Additionally, the Data Block Refill signal was asserted to make the processor handle both data and address cache misses in exactly the same manner, as quad word reads.

The remaining interrupt inputs, INT*[3..5], can be selected by the user at reset. This enables different functionality if the 50 MHz processors are swapped out for a slower frequency model. Since the 50 MHz R3081 is restricted to a one times clock input and half frequency bus operation, the Half-frequencyBus* and 1xClockEn* inputs, INT(3)* and INT(4)* are asserted during reset. Additionally, the Coherent DMA Enable is not asserted on INT(5)* because the implementation does not use DMA.

5. Test Connectors

Test connectors were designed into the schematic to allow for ease of troubleshooting. The initial design had connectors on each bus of the system. They were between each processor and the voter FPGAs, the voter FPGAs and the memory system, the FIFOs and the System Controller, and one collecting up all the control signals. Unfortunately, the Orcad tool used by the contractor to convert the schematics to a PCB Layout could not support the large number of pins in the system. The majority of the test connectors had to be pulled from the design. Only the connectors between the FIFOs and the System Controller remain in the implementation. For troubleshooting, chip clips will have to be used to extract the signals from the board.

D. MICROPROCESSOR AND ADDRESS LATCH

Although the system requires the support elements to function correctly, the key element in the system is the microprocessor. In the case of the TMR system, it is actually the three parallel microprocessors. This section discusses the model selection of the R3081 microprocessor. Since the hardware is being presented as it is grouped in the schematics, the address latches are also discussed here. The schematic diagrams of the three microprocessor/address latch pairs along with the rest of the system are given in Appendix A.

1. Microprocessor

Even though the type of processor for the system was determined in Ref. [9], the frequency and version of the model for the implementation still had to be selected. The simulations conducted in the Ref. [10] used a 40 MHz system clock with a half-frequency bus. In order to have the ability to replicate the simulation values and push the design envelope, a processor with the ability to handle frequencies of 40 MHz or higher was needed. The 50 MHz IDT 79R3081-50MJ was selected because it has an allowable frequency range of 40 to 50 MHz. This processor comes in two versions, the base and extended. The only difference between the two is the extended version adds a full-featured Memory Management Unit (MMU) to manage the virtual to physical address translation. Since the simulation used a base version and the processor is socketed for ease of future upgrades, the base version was selected for this implementation and the extended version was left for future inclusion. [Ref. 10]

2. Address Latch

In order to reduce the pin count of the R3081, IDT chose to multiplex the address and data bus together. This generates a requirement for the designer to separate the two buses using a latch. The processor generates an ALE signal to capture the address into the latch. Normally, a transparent latch is used, which allows the address to appear on the address bus a short propagation delay after ALE goes high. This gives the memory system more time to access the desired memory location.

In this design, the address latch that was simulated was a 74FCT373. The FCT family of logic uses fast CMOS as its core logic with TTL inputs and outputs. The high power consumption of the FCT logic family did not fit the design parameter of a future space flight board. This led to the selection of the 74AHCT573 for the implementation to take advantage of its low propagation delays and power requirements. [Ref. 17 and 18]

E. MEMORY SPACE

There are three types of devices found in the memory system of this design, Programmable Read Only Memory (PROM), Random Access Memory (RAM), and memory mapped I/O peripherals. This section will present an overview of the memory space supported by the R3081 and the incorporation of the PROM, RAM, and I/O peripherals into the memory space of the implementation.

1. R3081 Memory Space

The R3081 supports a 4 GByte memory space, which is broken down into four distinct virtual address areas: KUSEG, KSEG0, KSEG1, and KSEG2. The KUSEG, or kernel/user segment, is a cacheable 2047 MByte area used for both kernel and user processes. This gives the kernel access to user memory regions. KSEG0, or kernel segment zero, is a cacheable 512 MByte area that is always translated to the 512 MByte region of the physical address space beginning at address 0x0000_0000. Since this region is cacheable it is normally used for kernel executable code and some kernel data. KSEG1, or kernel segment one, is a non-cacheable 512 MByte area that is also mapped to the lowest 512 MByte area of physical memory. Since this section is non-cacheable, it is normally used for memory mapped I/O, boot ROM code, and operating system data areas. KSEG2, or kernel segment two, is a cacheable 1023 MByte area analogous to KUSEG, but it is only accessible from kernel mode. This area is normally used by operating systems for stacks, process data, and dynamically allocated data areas. A diagram of the base version of the virtual to physical address translation is shown in Figure 3.5. [Ref. 10]

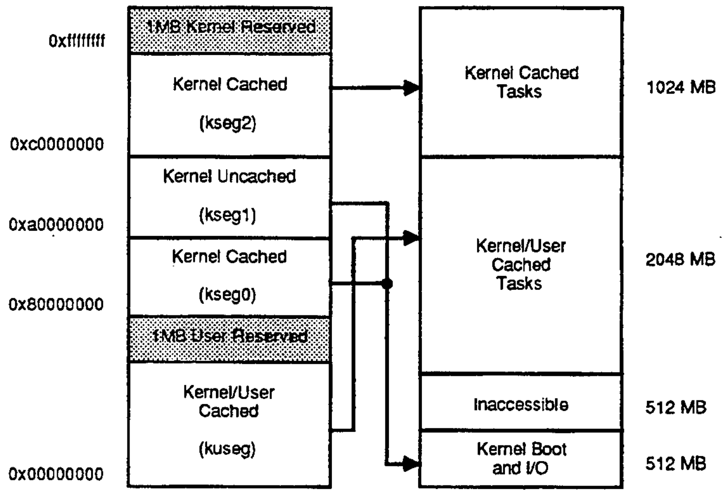


Figure 3.5. Virtual to Physical Address Translation. From Ref. [10]

2. PROM

Although the R3081 supports a 4 GByte memory space, there was no need to populate the entire space for this implementation. The driving factor for the PROM size and type for the implementation was the operating system (O/S). Since a set of boot PROMs containing VxWorks was part of the O/S purchase, the PROM space was designed around the VxWorks PROMs, which are standard 27C010 PROMs with 512 kBytes of storage and a 150 ns access time. This only required a minor change to the simulated PROM space from Ref. [9], which is the addition of two address lines to account for the increase from 128 kBytes to 512 kBytes.

In order to have the ability of running user defined boot programs, a set of Atmel 28C010-15 PROMs were ordered. The difference between the 28C010 and the 27C010 parts is that the 28s are electrically erasable, while the 27s need fifteen to twenty minutes under an ultra violet light to be erased, otherwise they are functionally equivalent. Because of long delivery times needed for the 28C010s, AMD 27C010-12s had to be substituted. These parts have a 120 ns access time versus the 150 ns access time of the VxWorks PROMs [Ref. 18]. Fortunately, the detailed timing diagrams presented in Chapter IV show that the memory controller state machine will support both 120 and 150 ns accesses speeds without the need for changes.

As far as their location within the 4 GByte memory space, the PROMs were physically located between addresses 0x1FC0_0000 and 0x1FC7_FFFF. The starting location was specified by the R3081 architecture, because 0x1FC0_0000 is the address vector loaded into the Program Counter during a reset [Ref. 10]. The schematic of the PROM space can be found in Appendix A.

3. SRAM

Since the PROM usually only maintains a small kernel of boot code, the processor needs a segment of SRAM to store the programs, applications and data that are needed during execution. The design of the SRAM space for the TMR system was based on two different factors: how large a memory space was needed and which segments of memory to populate. The following sections present the decisions leading to the final implementation of the SRAM space, which is shown in Appendix A.

a) Size

The memory space used in the simulated model of Ref. [9] was a 128 kByte block designed by placing four 32 k x 8 bit memory chips in parallel. This design proved more than sufficient for the simulations. In fact, only a small portion of the memory block had data placed in it. Because of the success during simulation, the memory space for the implementation started with the same design concept. The only question that remained was how many of the 128 kByte blocks would be required.

Because the Software Testbed function was expected to place a larger demand on the memory system, that model was used to determine the amount of SRAM needed for the implementation. In the Software Testbed model there are two contributing factors to the size of the memory: the O/S and user programs. The minimum SRAM requirement for a development system that includes all the standard VxWorks features is 2 MBytes [Ref. 19]. Because the implementation does not require the full complement of features, such as the networking capabilities, the O/S SRAM requirements are reduced to under 0.5 MByte. Allowing an additional 0.5 MByte for stacks and other dynamically allocated memory, the total memory requirement for the O/S is estimated to be approximately 1 MByte. Since the size of user programs can vary greatly, an additional 1.5 MByte of SRAM was included in the memory space. This brought the total SRAM requirement to 2.5 MByte.

Using the simulated model of 128 kByte memory blocks, it would take 20 different memory blocks to implement the 2.5 MByte memory array. Furthermore, each memory block would require its own chip select signal and it would take 80 chips to implement, which would consume a very large area on the printed circuit board. In order to reduce these requirements, a 128 k x 8 bit memory chip was selected. This reduced the chip select signals to 5 and the chip count to 20.

The IDT71024 CMOS Static RAM was selected for the implementation of the main memory. This chip was selected over other SRAM manufacturers because two other major components of the TMR system were also manufactured by IDT: the microprocessors and FIFOs. This was done to reduce the number of different manufacturers used and to increase interoperability. The initial specification called for a 20 ns read/write cycle to allow for faster bus speeds when using 40 MHz processors. Because the 20 ns SRAM was not available in a timely manner, the IDT71024S12 was used, which has a faster read/write cycle time of 12 ns [Ref. 20].

b) Memory Segments

As discussed in the beginning of the memory section, the R3081 supports four different virtual address segments. From the previous section, the design was determined to consist of five different 0.5 MByte segments. This section looks at how the five SRAM segments are used to populate the four virtual address segments.

Only two of the four virtual address segments are populated with SRAM. They are the KUSEG and KSEG0. Since KSEG0 is used for kernel code, one segment of the O/S SRAM was placed there to provide a random access location for the contents of the PROMs to be transferred to and executed from. The remaining O/S segment and the three user segments were placed in the KUSEG in one contiguous memory space. This was done to allow the O/S greater flexibility when assigning code to memory locations.

The other two virtual address segments, KSEG1 and KSEG2, are not populated with SRAM in this implementation. Since KSEG1 is uncacheable and used as the address area for the PROM and peripherals, there was no need to populate it with SRAM. Because KSEG2 is restricted to kernel use and has the same functionality as KUSEG, KSEG2 was not populated in this implementation. This area can be populated in later designs simply by changing the Address Voter FPGA's chip select equations, since the chip select signals are decoded from the voted upper address lines.

4. Input/Output (I/O)

Since the UART is a memory mapped peripheral, it seemed logical to discuss the design of the I/O interface in the Memory Space portion of this chapter. A peripheral is memory mapped if it is accessed by decoding an address placed on the bus. One of the design issues associated with the I/O system was selecting the address for its registers. Because the software development portion of this project had access to code mapping the

UART to 0xBFE0_0003 and that address was not used by any other peripheral, the UART was assigned that address for sake of continuity.

The I/O interface performs the important function of interfacing the TMR system to the HCI. Because the simulated model used the Verilog design suite to handle the interface, this portion of the system had to be designed from the ground up. The design of the interface consisted of selecting a type of interface and then implementing the selection.

a) Interface Type

There are many different types of computer system interfaces on the market today. When trying to determine which interface was best for this implementation of the TMR R3081, several factors were considered. They were the amount of data that needed to be transferred, how fast it needed to be transferred, and how far it needed to travel. This section looks at how these factors contributed to the selection of the interface for the TMR R3081 system.

In this design there are two major uses of the I/O interface. They are the transfer of the user program to the TMR System from the HCI and the transfer of the FIFO data to the HCI from the TMR System. Because radiation testing causes SEUs to be induced at a much higher rate than the space environment, the FIFO data transfer was considered to be the critical transfer. If the additional bus traffic before and after the interrupt is ignored and only the register writes to memory are read into the FIFOs, there is a requirement to transfer 140 register values from each processor [Ref. 10]. The 140 registers are broken down by location in the Table 3.2. The 140 registers contain 816 bytes of data. In order to transfer each byte of data there is normally an overhead of three additional bits for start, stop and parity. This brings the total of bits to transfer to 8,976 per interrupt for each processor. With three processors, the TMR system will need to transmit 26,928 bits to the HCI system for each vote error interrupt.

Register Location	Number of Registers	Bytes/Register	Registers*Bytes
CP0 Registers	10	4	40
GP Registers	32	4	128
FP Control	2	4	8
FP Registers	32	4	128
TLB	64	8	512
Total Bytes Per Processor To Transfer			816

Table 3.2. R3081 Internal Registers

The speed of the transfer is somewhat related to the size of the data element: the larger the data block to transfer, the higher you want the transfer rate. In this implementation the goal was to be able to transfer the FIFO data to the HCI in approximately 250 ms. This value was selected to allow four sets of interrupt data to be transferred per second. In order to transfer 26,928 bits in under 250 ms a minimum baud rate of 108 kbaud is required.

One of the most critical factors in choosing the interface was the distance the data had to travel. One effect of increasing the transmission distance is a lowered maximum frequency of the driver. This happens because the long communication lines add capacitance to the load, which draw large currents and induce noise on the lines [Ref. 21]. Due to the physical layout of the radiation chambers a transmission length of approximately fifteen meters is expected. This distance allows the testers to be shielded from the radiation beam during testing.

To summarize, the interface needed for the implementation should be able to transfer data near 108 kbaud over fifteen meters. Research into the different interfaces led to solutions ranging from an Electronic Industry Association (EIA) 232-F, also known as RS-232-F, to an Ethernet connection. In the end, the EIA-232-F was selected. Other than meeting all the criteria, this interface was selected because it was the easiest to understand and implement. This was important because the nature of this project has it passed on from student to student and anything that can reduce the amount of ramp-up time is a benefit. The actual implementation of the interface is presented in the following section.

b) I/O Interface Implementation

The I/O interface is implemented using a universal asynchronous receiver transmitter (UART) and a set of line drivers. The purpose of the UART is to convert parallel data from the TMR system to serial data for transmission or convert received serial data to parallel data for use by the TMR system. It also generates the control signals used to establish communication between the two devices. The line drivers convert the serial data and control signals from digital values to signals capable of being transmitted across the serial cable. The selection of these two devices is presented in this section and their general interconnections are shown in Figure 3.6. The design schematics are located in Appendix A.

The selection of the UART turned out to be one of the easiest part selections involved in the implementation. The Texas Instruments TL16C750 was selected for the implementation for its maximum transfer rate of 1 Mbaud and its ability to meet the EIA-232 standards [Ref. 22]. Additionally, it had 64 byte FIFO buffers for both the transmitter and receiver, which allowed the processor more flexibility in reading and writing to the UART. It also has a Programmable Baud Rate Generator to allow different baud rates to be used for interfacing. Unfortunately, this part was not available without a lengthy delay, so the TL16C550C was substituted. This part was completely functional and pin compatible with the TL16C750, but only has 16 byte FIFO buffers. [Refs. 22 and 23].

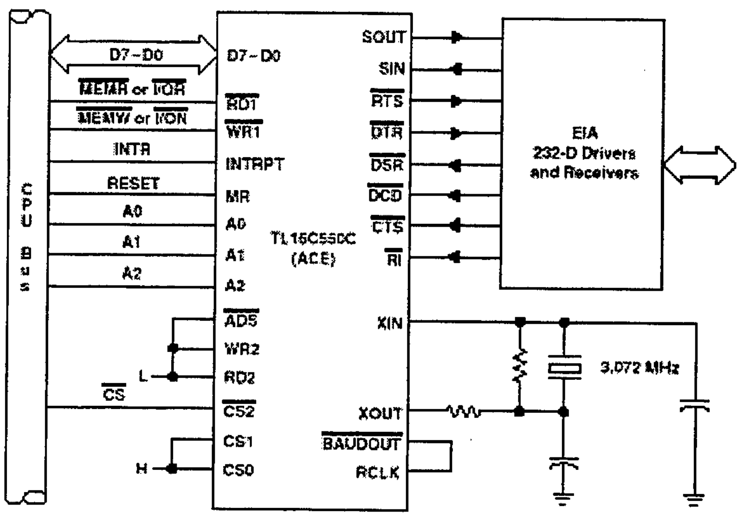


Figure 3.6. Bus/UART/Line Driver Interconnections. From Ref. [22]

Since the line drivers are the devices that adhere to the transmission standards, they became the most important aspect of the I/O interface. The initial set of drivers selected was the Texas Instruments MAX3243. Because very few high end line drivers are available, it was very easy to make this part choice, but they were also unavailable for delivery. The Maxim MAX239 was then substituted in its place. This device meets all the requirements of the EIA-232-E standard and operates at baud rates of up to 120 kbps [Ref. 24]. This allows data to be transferred at the maximum allowed rate of the HCI at this time.

Two important notes about the I/O interface are that if the HCI gets upgraded during follow on work to allow higher data transfer rates, the line drivers are socketed to allow for easy exchange. Also, the UART and line driver pair are implemented twice in the TMR R3081 system. One set is used for the Data Port and the other set is used for the Control Port.

F. FIFO INTERFACE

The FIFO interface is used to collect data off the three microprocessors' A/D busses during vote error interrupts. This is done by "snooping" the address and data information off the bus during the interrupt handler. The data contains each of the microprocessors' internal register sets, which are transferred out the Control Port to the HCI for analysis. This section describes the design of the FIFO interface. A schematic diagram of the FIFO array is given in Appendix A.

1. Control Bus Design

The simulated system of Ref. [9] used two 1k x 18 bit FIFO buffers in parallel to collect the address, data, and control information from the busses of each processor. The data was then read out of the FIFOs using the System Controllers 104 bit wide System

Interface. The interface of the FIFOs and the UART, which replaced the System Interface in the implementation, created the design problem, because the 8 bit wide UART bus did not match up with the 18 bit wide FIFO bus. Two solutions for this problem came to mind: add glue logic between the FIFO and UART or redesign the interface. The later solution won out. By replacing the two 1k x 18 bit FIFOs with five 1k x 8 FIFOs the interface problem quickly disappeared. The first four FIFOs are used to collect the address and data information, while the fifth one collects all the control information. Since the FIFOs use tri-state outputs, all fifteen of the 8 bit wide FIFO output busses can be placed on the same bus as the 8 bit wide UART bus, which provides a streamlined connectivity to the UART.

2. FIFO Selection

For reasons discussed in the SRAM section, FIFOs manufactured by IDT were selected for the implementation. The large number of FIFOs to choose from allowed plenty of flexibility in the selection of the FIFO for the implementation. Since the simulated version of this design did not need to handle a large number of interrupts, the FIFO buffer could be relatively shallow. For the implementation, the design had to ensure enough space in the FIFO to prevent the FIFO from overflowing and losing data. By doubling the size of the buffer to a depth of 2,048 (2k), the system can support up to seven errors without overflow, assuming no data is transferred out. Because both the address and data bus contents are written into the FIFO for each register write, the 140 registers require 280 positions in the FIFO for each interrupt. Unfortunately, the 2k FIFO was not available and the 4k FIFO had to be substituted in its place, which removed all fears of overflow. The part number for this device is IDT72240L15TP and its specifications can be found in Ref. [25].

This chapter has presented the design of the hardware used in the implementation of the TMR system. In that discussion three very important pieces were omitted: the design of the System Controller, Voters, and Memory Controller. One common element all three of these devices share is that they are designed in programmable logic devices, either FPGAs or PLDs. Because their design is more of a software implementation, they are presented together in the following chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PROGRAMMABLE LOGIC DESIGN

Because programmable logic offers flexibility to the system designer, it was selected to perform several key functions in the TMR R3081 design. First, the Memory Controller and Memory Enable functions were implemented in Programmable Logic Devices (PLDs). The selection process used for the PLDs and a description of the signals they produce are presented in Section A of this chapter. Field programmable gate arrays (FPGAs) were used to implement the voting logic, address decoding, timer, and System Controller functions. The selection process and the design of the three FPGAs used in the system are presented in Section B of this chapter. Section C ends the chapter with a detailed timing analysis of the processor interface with all the peripheral devices that interact directly with the processors. The programmable logic components of the design are identified as shaded blocks in Figure 4.1.

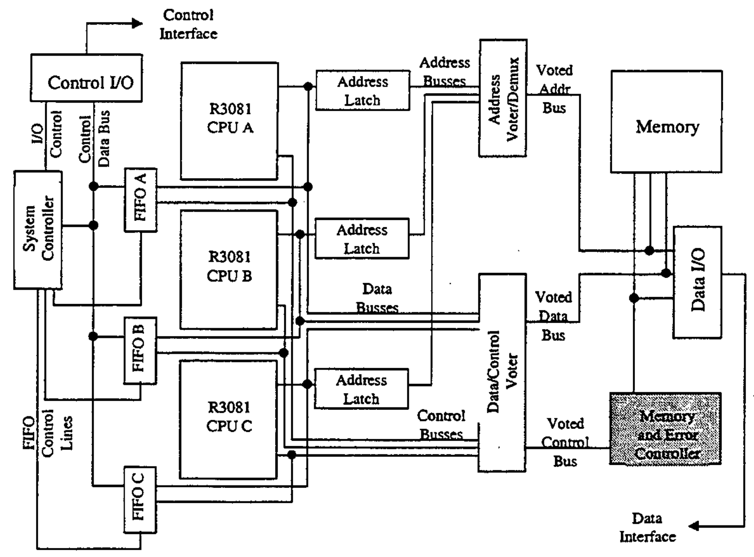


Figure 4.1. Programmable Logic Device Identifier

A. PLD DESIGN

The simulated system design presented in Ref. [9] used three user-defined modules to perform the Address Decoder, Memory Controller, and Memory Enable functions. Those designs were based on PLDs presented in Ref. [11], which is an Integrated Device Technology (IDT) system design example using an IDT R3051 microprocessor. Although this is an older processor with fewer capabilities, the backward compatibility of the R3081 made it pertinent to this design. Because of its applicability, it was also used as a design template for the implementation designed in this thesis.

The design example in Ref. [11] used three PAL22V10s to perform the Address Decoder, Memory Controller, and Memory Enable functions. Because of the number of inputs and outputs required for each of these functions, the smallest PLD that they would fit in was a PAL22V10. The FPGAs did offer one other option to the implementation of these three functions. Since the two Voter FPGAs generate the majority of the input signals used by these PLDs, the functions could have been incorporated into one or both of the Voter FPGAs. It was decided to incorporate the Address Decoder into the Address Voter FPGA for the following reasons:

- All the signals required to perform the address decoding were present in the Address Voter FPGA.
- The FPGA had enough available I/O pins to accommodate the chip select signals.
- The functions were similar enough that their consolidation did not obscure any integral functionality of the system.
- The combination of the two functions streamlined the overall design.

Because the Address Decoder was incorporated into an FPGA, its design is presented later in the chapter. Although the Memory Controller and Memory Enable functions met most of the criteria used to incorporate the Address Decoder, they were left for implementation in PLDs. The main reason was to provide better visibility of their functionality for future changes or upgrades.

The next step in the process was to select the PLD to be used in the design. Since the design example used 22V10s to implement their functions, it was also selected for the implementation in order to facilitate continuity. Because the 22V10 is a standard part that is fabricated by many companies, the specific manufacturer was not as important as the timing constraints of the available devices. The Atmel ATF22V10C-7PC was selected for use in the implementation. This electrically erasable CMOS device offered a maximum clock to output delay from registered outputs of only 2.5 ns and a maximum propagation delay of 7.5 ns [Ref. 26].

The next issue faced in the design of the PLDs was how to actually program them. Research led to a software program called WinCUPL, which is a Windows version of Universal Compiler for Programmable Logic (CUPL). Like other programmable logic compilers, CUPL defines its own language and syntax requirements for device design. Designs may be implemented as logic equations, truth tables, or state machines and then simulated using the software tools that are provided. The two most important factors leading to the selection of CUPL as the design environment was that it is produced by Logical Devices, Inc. (LDI) and that it has the ability to design FPGAs, Xilinx FPGAs in particular. The connection to LDI was important because the device programmer being used is also distributed by LDI. This hardware/software combination provides good interoperability and is actually referenced in the CUPL Users Guide [Ref. 27]. For reasons discussed in the FPGA Design Section of this chapter, the ability to program Xilinx FPGAs was also thought to be important.

The schematics showing how the Memory Controller and Memory Enable PLDs are interconnected within the TMR system are provided in Appendix A and the CUPL design and simulation files for the PLDs are provided in Appendix B. The next two subsections describe the two devices.

1. Memory Controller PLD

The Memory Controller PLD acts as the main piece of glue logic between the microprocessors and the memory space. When the microprocessors generate the signals initiating a bus transaction, the Data and Control Voter FPGA perform a majority vote on them and pass them on to the Memory Controller PLD. The Memory Controller uses those signals to generate the signals required by the memory space to perform the bus cycle and the signals the processor requires to terminate the bus cycle. This section describes the signals generated by the Memory Controller PLD. A diagram of the PLD with its inputs and outputs is given in Figure 4.2 and its program file is in Section 1 of Appendix B.

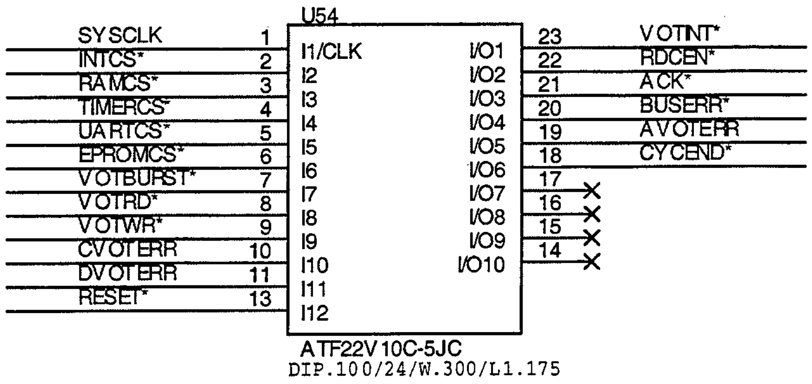


Figure 4.2. Memory Controller PLD

a) Counter

The Memory Controller PLD uses an internal four-bit counter to provide a timing reference for the different bus cycles. This allows the processors to interact with devices of various speeds by introducing wait states into the bus cycle for devices that can not respond in a single access time. Also, because the R3081 supports burst reads, which is a read of four consecutive memory locations in a single read cycle, the counter also provides a reference for each of the four memory accesses.

The counter is implemented in the PLD by using pins I/O[7..10] for internal interconnect, which is why they are shown as unconnected in Figure 4.2. It uses the RESET*, CYCEND*, SYSCLK, VOTRD*, and VOTWR* inputs to determine when to count and when to reset back to zero. The RESET* signal originates from the System Reset signal and is used to reset the counter whenever a system reset is required. The CYCEND* signal is another internally generated signal used to identify the end of the current bus cycle and to reset the counter back to zero. Its implementation is discussed next. The VOTRD* and VOTWR* signals are the majority voted RD* and WR* signals generated by the microprocessors to initiate the bus cycle. Unless the RESET* or CYCEND* inputs are asserted, the counter increments on each rising edge of SYSCLK after the VOTRD* or VOTWR* signal has been asserted. It will continue to count until either the CYCEND* or RESET* inputs are asserted.

b) Cycle End (CYCEND)*

The CYCEND* signal is used to signal the end of the current bus cycle. Since all the signals generated by the Memory Controller and Enable PLDs are registered, they require a signal that will cause them to de-assert at the end of each bus cycle. This function could be performed by using the negation of the VOTRD* or VOTWR* signal, but it is not because they occur within the setup and hold times of the buffered/inverted SYSCLK* signal [Ref. 11].

The CYCEND* signal is generated from the RAMCS*, EPROMCS*, UARTCS*, TIMERCS*, INTCS*, VOTRD*, VOTWR*, VOTBURST* inputs and the counter contents. The timing of the CYCEND* signal is dependant on the latency required to access the device and the type of bus cycle that is being run. The five active low chip select signals (those that end in CS*) and the counter contents are used to determine the device latency. The VOTRD*, VOTWR*, and VOTBURST* signals are used to determine the type of bus cycle. For this implementation, all the devices could perform a single word read or write within a single access time, except for the EPROM, which required one additional wait state for the read cycle and for which the write cycle is not applicable. Quad word reads are only applicable to the SRAM and EPROM and are detected by the VOTBURST* signal being asserted. During this type of read, the

CYCEND* signal is not asserted until the fourth word has been accessed, which takes six clock cycles for SRAM and seven clock cycles for EPROM.

c) Read Buffer Clock Enable (RDCEN)*

The RDCEN* signal indicates to the microprocessors that the memory system has sufficient time to process the read request and will have valid data on the A/D bus when the microprocessor is ready to latch the data in. Because this signal has to be asserted prior to the rising edge of SYSCLK* for the data to be latched in on the next falling edge, valid data does not have to be on the bus when RDCEN* is asserted, only prior to the next falling edge of SYSCLK*. During quad word reads, RDCEN* must be asserted and de-asserted four times. Along with enabling the latch process, the RDCEN* signal also serves to terminate the bus cycle, which will be explained with the ACK* signal.

The RDCEN* signal is generated by the five CS* signals, VOTRD*, VOTBURST*, and the counter. As the name implies, this signal is only asserted on read transactions. With exception of the EPROM, the remainder of the memory system can perform single word reads in a single clock cycle. Because of that, RDCEN* is asserted with the rising edge of SYSCLK while the counter contents is 0x0. The EPROM requires one wait state, therefore, RDCEN* is asserted on the following clock when the counter contents is 0x1. During the burst read cycles, RDCEN* is asserted four times for each bus transaction. Because the Memory Controller uses SYSCLK as the reference for the state machine, the signal is asserted for one clock cycle and then de-asserted for the next one. Since the SRAM does not require any wait states, RDCEN* is asserted during the clock cycles when the counter contents are 0x0, 0x2, 0x4, and 0x6. The EPROM requires a single wait state in the initial access time, but the remaining reads can be completed in the time required to toggle the RDCEN* signal from on to off and back to on. Therefore, it is asserted when the counter contents are 0x1, 0x3, 0x5, and 0x7 for EPROM burst reads.

d) Acknowledge (ACK)*

The ACK* signal is sent to the microprocessor to indicate that the memory system has processed the bus cycle sufficiently. The processor may then either move on to the next write buffer or release the internal core to process the read data. On both single word reads and burst reads, the microprocessor can implicitly generate ACK* and terminate the bus cycle based on the RDCEN* signal, although this technique will degrade performance on burst reads. To optimize performance, the ACK* signal was generated by the Memory Controller for the burst read in this implementation. On writes, the microprocessor uses the ACK* signal generated by the Memory Controller to terminate the bus cycle. This specifies that the memory system has completed the transaction and that the processor can stop driving the A/D bus with the data.

As with the other signals looked at, ACK* is generated by the five chip select signals, VOTRD*, VOTWR*, and VOTBURST* inputs and the counter contents. The ACK* signal for single word reads was not explicitly generated as discussed before. The burst read ACK*, which improved performance by releasing the processing core to use the data already in the read buffer, must not be generated more that four clock cycles prior to the end of the transaction. This ensures that the last word of the read is in the buffer by the time the execution core expects it to be there. This signal is generated when the counter contents are 0x3 for SRAM reads and 0x4 for EPROM burst reads. This signal must be explicitly generated on all writes. Since the EPROM can not be written to, all acknowledges for write transactions occur without wait states or when the counter contents are 0x0.

e) Bus Error (BUSERR)*

The BUSERR* signal is generated by the memory system to inform the microprocessors that they have tried to access an invalid address. The BUSERR* signal uses the counter to determine when this happens. The counter starts counting when a VOTRD* or VOTWR* signal is asserted by the processors to signify the start of a bus transaction. After initiating the bus cycle, the processor then waits for a RDCEN* or ACK* signal from the memory system. These signals will never be generated, because they require a valid address to generate the chip select signal that they are dependant on. The counter will continue to count until it reaches 0xF, which causes the BUSERR* signal to be asserted.

f) Voter Interrupt (VOTINT)*

The VOTINT* signal is generated by the Memory Controller to indicate that a miss-compare has happened in one of the Voter FPGAs. This signal is routed to the INT3 input of the three microprocessors and held asserted until the processors acknowledge it during the interrupt handling routine. This signal uses the AVOTERR, CVOTERR, DVOTERR, and INTCS* inputs. The signal is asserted when any of the xVOTERR signals are asserted. When the interrupt handler performs a write to the

Interrupt Acknowledge Address, the INTCS* signal is generated and de-asserts the VOTINT* signal.

2. Memory Enable PLD

The Memory Enable PLD supports the Memory Control PLD during bus transactions. It does this by generating the read and write enable strobes for the memory system, read and write data enable signals to control the drivers on chips between the busses, and a positive and negative logic synchronous reset. The diagram of the Memory Enable PLD with its inputs and outputs is given in Figure 4.3 and its program file is in Section 3 of Appendix B.

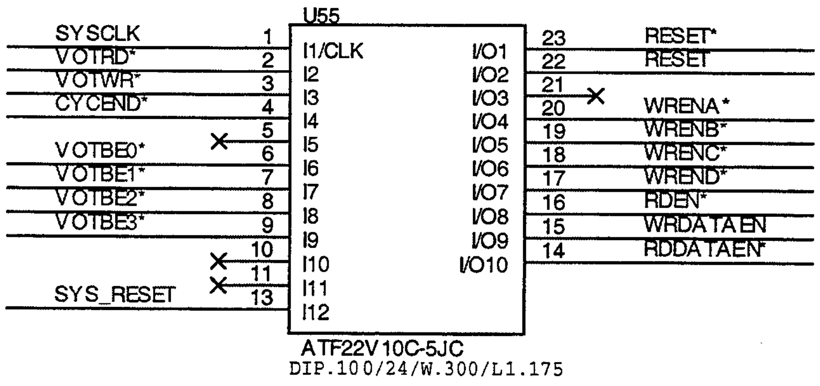


Figure 4.3. Memory Enable PLD

a) Read Enable (RDEN)*

The RDEN* signal works in conjunction with the chip select signal, which is generated by the Address Voter FPGA, to activate the output drivers of a memory device. Because all the logic between the processors and the chip select signal is combinatorial, the possibility exists for the wrong chip-select signal to 'glitch' while the Address Bus is settling. In order to avoid the wrong address being accessed during this time, the RDEN* signal is held high until later in the bus transaction. This allows the chip-select signal to stabilize and gives the memory device time to access the address location and move the data requested to its outputs. The RDEN* signal is then used to activate the device's output drivers in order to drive the Data Bus. Unlike the write enable signals, only one RDEN* signal is required. Since reads are always done as complete words, four bytes wide, there is no reason to select which bytes will participate in the read. The RDEN* signal is just passed to all four devices in the memory block in parallel.

The RDEN* signal is generated using the SYS_RESET*, CYCEND*, and VOTRD* inputs. The RDEN* is asserted on the next rising edge of SYSCLK after VOTRD* is asserted. The delay is used to allow the chip select signals to settle. The signal is de-asserted by CYCEND* being asserted. This turns the output drivers of the memory device off as early as possible in the bus transaction and helps reduce the possibility of bus contention. The SYS_RESET* input only effects the RDEN* signal during resets, at which time it de-asserts the signal if it was asserted.

b) Write Enables

Since the R3081 supports byte, halfword, tri-byte and full word writes, the Write Enable strobes are used to determine which bytes of the word are going to be involved in the write transaction. The four write enable signals that are produced are WRENA*, WRENB*, WRENC*, and WREND*. WREND* corresponds to the most significant byte and WRENA* corresponds to the least significant byte.

The Write Enable strobes are generated by the SYS_RESET*, VOTWR*, CYCEND*, VOTBE0*, VOTBE1*, VOTBE2*, and VOTBE3* signals. The CYCEND* and SYS_RESET* signals are used in the same manner for the Write Enables as the RDEN* signal. Each of the Write Enable strobes corresponds to a single voted Byte Enable signal from the microprocessors. When the VOTWR* signal is asserted, the Write Enable strobe corresponding to each of the asserted Voted Byte Enable signals will also be asserted. Additionally, the R3081 supports byte, half word, tri-byte, and full word writes. To clarify the point, the truth table for this function is given in Table 4.1. Because the signals are negative logic, the 0s signify an asserted signal.

VOTWR*	VOTBE[3..0]	WREND*	WRENC*	WRENB*	WRENA*
0	1110	1	1	1	0
0	1101	1	1	0	1
0	1011	1	0	1	1
0	0111	0	1	1	1
0	1100	1	1	0	0
0	0011	0	0	1	1
0	1000	1	0	0	0
0	0001	0	0	0	1
0	0000	0	0	0	0

Table 4.1. Write Enable Assertion Table

c) *Data Enables*

The Read Data Enable (RDDATAEN) and Write Data Enable (WRDATAEN*) signals are used to control the output drivers on devices separating different data busses in the system. In this system, the Data Voter FPGA separates the three microprocessor's A/D Busses from the VDATA Bus and the driver/transceiver separates the VDATA Bus from the memory system's DATA Bus. This has to be done to prevent bus contention when the processor turns the bus at the end of a read transaction after a device with a slow turn off time has been accessed. The faster turn off time of the FPGA and driver/transceiver isolate the slower device until the bus is cleared for the next transaction.

The WRDATAEN is a positive logic signal generated by SYS_RESET*, VOTWR*, and CYCEND*. It allows data to flow from the processors, through the Data Voter, and into the memory system. As with the RDEN* signal, the SYS_RESET* clears the signal on reset. The VOTWR* is used to assert the signal and CYCEND* is used to de-assert it. By using the CYCEND* to de-assert the strobe, the devices are tri-stated earlier in the cycle, which allows them more time to turn completely off before the processor drives the bus with the next address. The RDDATAEN* strobe works exactly the same as the WRDATAEN strobe, except it is a negative logic strobe and uses the VOTRD* signal instead of the VOTWR*.

d) Synchronous Resets

Since some of the peripherals in the Memory System required positive logic resets and some required negative logic resets, a reset signal had to be generated for both polarities. Although this could have been done by placing an inverter on the SYS_RESET* signal to get the positive logic reset, it was implemented in the PLD. This solution provided a common location for both the positive and negative logic synchronous reset signals. These signals are just registered and registered/inverted copies of the SYS_RESET* signal.

Although the Memory Controller and Memory Enable PLDs are key elements of the glue logic in this system, they are not the only programmable devices used. The next section of this chapter presents the selection and design of the three FPGAs used in this system.

B. FPGA DESIGN

Although the simulated system in Ref. [9] used three user-defined modules to perform the voter functions and the Verilog Design Suite to perform the System Controller function, they were implemented in three FPGAs in this design. This section of Chapter IV presents the FPGA selection process and the design of the three FPGAs used in the system. Schematics of the FPGA designs are presented in Appendix C.

1. FPGA Selection

Unlike the PLD selection, the FPGA selection process required more research. The initial focus of the FPGA search was trying to find an FPGA vendor that manufactured the same FPGAs in both commercial and rad-hard versions. This was important in maintaining the design goal of implementing a system that could easily be converted for future space use. If the FPGA selected for the implementation was available in both commercial and rad-hard versions, the upgrade to a space flight model could be completed easily. The programming files designed for the commercial model could also be used to program the rad-hard FPGA. This search criterion narrowed the possible vendors down to Actel and Xilinx.

The next factor used to narrow down the search was FPGA size. Since FPGAs are a very powerful implementation, the limiting factor for this design was not based on internal logic, but on the number of user definable I/O pins. The FPGA had to be large enough to hold a complete functional unit. That is, a complete 32-bit voter needed to fit in a single FPGA. This required a minimum of 130 user definable I/O pins: three 32-bit inputs, one 32-bit output, and at least two control lines. Actel had one rad-hard FPGA that met the requirements with 140 user definable I/O pins and Xilinx had three with 192, 288, and 384 user definable I/O pins.

Although any one of the four FPGAs listed in the previous paragraph would have worked in the implementation, a selection had to be made for the implementation. The Xilinx FPGAs offered one advantage over the Actel FPGAs in the way that they were programmed. Since the programming data is actually fused into the Actel FPGAs, they would have to be physically taken off the PCB and reprogrammed whenever changes in the design had to be made. The Xilinx FPGAs use a serial PROM (SPROM) to store the programming data. Whenever the system is reset, the program data is restored into the FPGA. By using the Xilinx FPGAs in the design phase, the 20-pin SPROM is removed for reprogramming, not the 240-pin FPGA. Although this may not sound like much of an advantage, the large number, small feature size, and close spacing of the FPGA pins would make its removal and replacement a very difficult task. This narrowed the viable options down to the Xilinx FPGAs.

The remainder of the selection process came down to how the implementation could best be packed in the FPGAs. The total pin count for the entire FPGA solution was 383 I/O pins. This would have actually fit in the 384 pin FPGA. This option was not taken because it did not leave enough room for design changes and future upgrades. The first design used the 384 pin FPGA to perform the voting function and the 192 pin FPGA to perform the System Controller function. When minimum order quantity requirements and cost were factored in, the final design used three 192 pin FPGAs. The System Controller, Address Voter and Decoder, and Data and Control Voter were each implemented in separate FPGAs.

Once the FPGA selection process was completed, the next consideration was how to program them. The software tool used to program the PLDs, WinCUPL, does offer a method for writing, compiling, and converting designs for use with Xilinx FPGAs. The only drawbacks were that the designs had to be entered in a hardware description language (HDL) and the conversion process was rather complicated. Because of this, other options were looked into for the FPGA design. It turned out that the new text book being used for the logic design course at NPS, Ref. [28], came with a student edition of the Xilinx Foundation software. Xilinx designed this software for the purpose of programming Xilinx FPGAs. After determining that it would support the FPGA selected for the implementation, it was selected as the design medium for the FPGAs. This software allows the design to be implemented in HDL, State Machine Design, Schematic Representation, or any combination thereof [Ref. 29].

Since the majority of the voter functions are combinatorial, the decision was made to design the Address and Data Voter FPGAs using the schematic representation. Their designs are presented in the next two sections. Although the actual implementation of the System Controller was not completed, its design is presented in Subsection 4 of this chapter.

2. Address Voter, Memory Decoder, and Timer FPGA

The first FPGA implemented was the Address Voter FPGA. Its functions are performing the majority vote on the three 30-bit address busses, decoding the upper thirteen bits of the voted address lines into chip select signals, and providing a system timer. The logic design of these functions is presented in this section and their schematic diagrams are provided in Section 1 of Appendix C.

a) Address Voter

Since the R3081 multiplexes its data and address pins, demultiplexers must be used to separate the Data and Address busses. The A/D bus of the microprocessor is split into two different paths, one registered and one not. The unregistered path is the Data Bus and the registered path is the Address Bus. When the R3081 initiates a bus cycle, the A/D bus is driven with the address to be accessed. The R3081 then asserts the Address Latch Enable (ALE) strobe and the transparent latches of the demultiplexer drive the Address Bus with the address to be accessed. The negation of ALE latches the address off the A/D bus into the demultiplexer, where it is held on the Address Bus until the following bus cycle when ALE is asserted again. Since all three of the microprocessors do this in parallel, the three CPU Address Busses must be majority voted prior to being passed to the Memory System.

The vote function for the Address Bus was designed hierarchically using Macros, which is a term Xilinx uses to describe the building blocks or sub-functions used to create an overarching design. First, a 3-bit Majority Voter/Error Detector, which is presented in Figure 4.4, was designed. This module takes three inputs and performs a majority vote on them and passes the output to the zero input of a multiplexer. The DATAA input is directly connected to the other input of the multiplexer. The Force input is connected to the control input of the multiplexer. When Force is not asserted, the multiplexer passes the voted data to the circuit output. When Force is asserted, the multiplexer passes the DATAA input to the circuit output. The circuit also uses a three input AND and a three input NOR, which are ORed together to generate a negative logic error signal when any of the input bits differ from the other two.

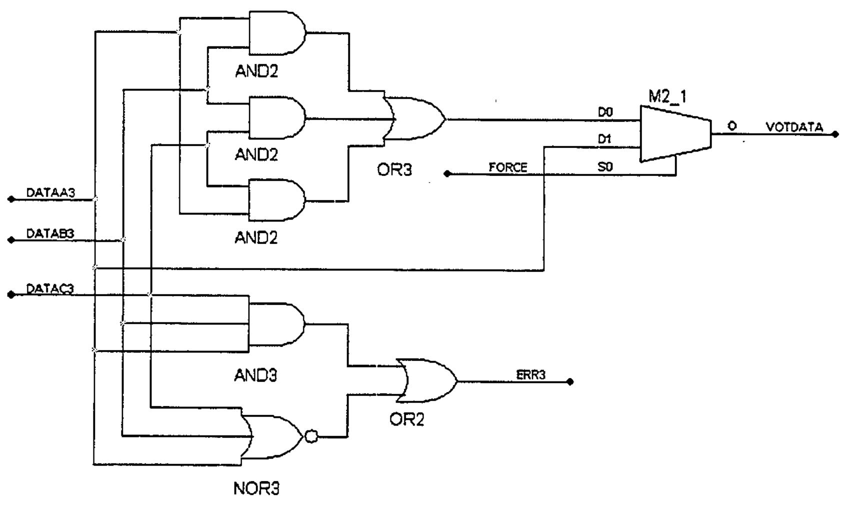


Figure 4.4. 3-Bit Majority Voter/Error Detector

The next step was to combine four of the 3-bit Majority Voter/Error Detectors into a macro called VOTER4. The schematic for the VOTER4 macro is given in Appendix C. This macro takes three four-bit wide busses and a Force input and produces two four-bit wide busses, a majority voted output bus and an error bus. The Force input causes the data on the DATAA Bus to be forced onto the OUT Bus through the multiplexer, which bypasses the vote logic.

Because the R3081 uses the four Byte Enable signals to select which of the four bytes are participating in the bus cycle, the Address Voter only needs vote the remaining 30 bits of the bus, because the Byte Enable signals are voted by the Control Voter. The Address Voter takes the three 30-bit Address Busses and a Force input to generate a 30-bit Voted Address Bus and a 30-bit Error Bus. It was implemented by placing eight of the four-bit wide voters in parallel, the two most significant bits are not used. The 30-bit Error Bus had to be reduced down to a single output, which was done by ORing the signals together. Since the Error signals are active low and the output is active high, the operation turned out to be a logical NAND. The Address Vote Error (AVOTERR) signal was then passed to the appropriate output pin. The Address Voter also separated the Voted Address Bus into two sections, the lower seventeen bits and the upper thirteen bits. The lower seventeen bits are sent to the output pins to make the VOTADDR[2..18] Bus, which is passed on to the Memory System for decoding. The upper thirteen bits are passed on to the Memory Decoder, which is presented in the following section.

b) Memory Decoder

The Memory Decoder takes the upper thirteen bits of the Voted Address Bus from the Address Voter and decodes them into chip select signals. In order to prevent multiple addresses being mapped to the same memory location, a full address decoding scheme was implemented, which means all thirteen of the upper address bits were used to generate the chip select signals. A chip select signal is generated for each of the five SRAM memory blocks, the EPROM, the UART, the Timer, and the Interrupt

Acknowledge. Table 4.2 shows the chip select name, its physical address, its virtual address, and the memory segment it belongs to. The addresses correspond to the thirteen bits of VOTADDR[32..19] with three zero bits concatenated to the end. The table shows that the Kernel SRAM resides in KSEG1, the remainder of the SRAM resides in the KUSEG, and all of the peripherals and the EPROM reside in the uncached KSEG1 as discussed in Chapter III. The Memory Decoder schematic is presented in Section 1 of Appendix C.

Signal Name	Peripheral	Physical Address	Virtual Address	Memory Segment
KRAMCS*	SRAM	0x0000	0x8000	KSEG1
URAM0CS*	SRAM	0x4000	0x0000	KUSEG
URAM1CS*	SRAM	0x4008	0x0008	KUSEG
URAM2CS*	SRAM	0x4010	0x0010	KUSEG
URAM3CS*	SRAM	0x4018	0x0018	KUSEG
TIMERCS*	TIMER	0x0440	0xA440	KSEG1
INTCS*	INTERRUPT	0x1000	0xB000	KSEG1
EPROMCS*	EPROM	0x1FC0	0xBFC0	KSEG1
UARTCS*	UART	0x1FE0	0xBFE0	KSEG1

Table 4.2. Chip Select Memory Map

c) Timer

The implementation of the timer in this design turned out to be one of the most time consuming problems faced. The initial plan was to create the timer using an integrated circuit designed specifically as a timing device. Unfortunately, neither of the two timers selected could be found, because both had become obsolete. Because VxWorks requires a system timer to function properly [Ref. 19], the timer had to be implemented in one of the FPGAs.

The sole purpose of the timer is to interrupt the processor at predetermined time intervals. In response to the interrupt, the operating system will update the system clock. In order to prevent the system from spending too much time processing timer interrupts, the goal was to keep the number of interrupts under 100 per second, but as close to a whole number as possible.

Since SYSCLK is used by the timer and is operating at 10 MHz, 40 interrupts per second would require the counter to count to 250,000 before asserting the interrupt. Since 250,000 is closer to 2^{18} than 2^{17}, the design was easier to implement with a 19-bit counter. By using 2^{18} ANDed with 2^{10} as the reset and interrupt signal, the counter will count to 263, 168 and then assert the interrupt. This works out to 38 interrupts per second.

To provide this functionality, a 19-bit counter macro was designed out of Toggle flip-flops. The basic counter was designed by running the SYSCLK signal to the first flip-flop's clock input. The output of the flip-flop was inverted and tied to the next flip-flop's clock input. This was done repeatedly until nineteen flip-flops were in series. The Timer schematic is provided in Section 1 of Appendix C.

Although this provided the basis for a timer circuit, certain functionality had to be added to make it useful to the system, such as a Timer Enable, Timer Disable, Timer Interrupt Enable, Timer Interrupt Disable, and a Timer Interrupt Acknowledge. The system allows the Timer Mode to be changed by performing a dummy write to a specific set of addresses. Table 4.3 lists the Timer Mode and its associated address.

Memory Address (Hexadecimal)	Timer Function
0x04400000	Timer Interrupt Acknowledge
0x04400004	Timer Enable
0x04400008	Timer Disable
0x0440000C	Timer Interrupt Enable
0x04400010	Timer Interrupt Disable

Table 4.3. Memory Mapped Timer Modes

This functionality was designed into the system using the VOTADDR[17..2] Bus, the TIMERCS* signal, three D flip-flops, and some combinatorial logic. The sixteen bits of VOTADDR[17..2] are fully decoded to provide five signals representing each of the desired functions: TIMERINTACK, TIMEREN, TIMERDIS, TIMERINTEN, and TIMERINTDIS. Figure 4.5 provides a schematic of the Timer functionality design.

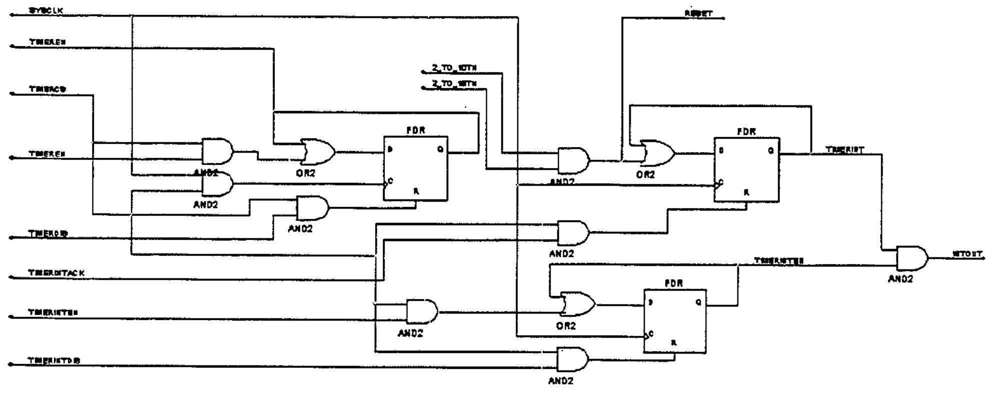


Figure 4.5. Timer Enable/Disable/Interrupt Design

The TIMEREN signal is ANDed with an inverted TIMERCS* signal and routed to the D input of the Timer Enable flip-flop. The TIMERDIS signal is ANDed with the inverted TIMERCS* and routed to the reset input of the Timer Enable flip-flop. The output of this flip-flop is routed to all of the Clock Enable and Toggle Enable inputs of the T flip-flops. Whenever the Timer Enable flip flop is set, the Timer will count. If the flip-flop is reset, the Timer will sit idle. To set the Timer Enable flip-flop, a write transaction is performed to address 0x04400004. This asserts the TIMERCS* and TIMEREN signals, which puts a one on the input to the Timer Enable flip-flop. The next rising edge of SYSCLK latches the data into the flip-flop and enables the timer. The output of the flip-flop is fed back to the input to hold the enable after the bus cycle has ended. The same method is used for the Timer Disable, except the signal is routed to the flip-flop's reset input and is asserted when a write is done to address 0x04400008.

When the counter reaches 263,168, the reset and interrupt signal is asserted. It is routed to the D input of the Interrupt flip-flop and the reset inputs of all the T flip-flops. On the next rising edge of SYSCLK, the counter is reset to zero and the Interrupt flip-flop is set. When the processor acknowledges the interrupt by writing to address 0x04400000, the inverted INTCS* is ANDed with the TIMERINTACK signal and routed to the reset input of the Interrupt flip-flop. The next rising edge of SYSCLK resets the Interrupt flip-flop and de-asserts the interrupt.

The output of the Interrupt flip-flop is ANDed with the output of the Interrupt Enable flip-flop. If the Interrupt Enable flip-flop is not set, the zero it supplies to the AND gate prevents the interrupt signal from being passed out of the FPGA. The Timer Interrupt Enable flip-flop works exactly like the Timer Enable flip-flop, except the TIMERINTEN and TIMERINTDIS signals are used when writes are done to addresses 0x04400010 and 0x04400018, respectively.

3. Control and Data Voter FPGA

Although the Control and Data Voter FPGA was implemented second, it was actually the easier FPGA to design. The main reason for that was that the Voter/Error Detector was already designed and just needed to be imported into this design. The only new designs needed were the Bidirectional Transceiver part of the Data Voter and the Synchronization Signal. The design of the Control Voter, Data Voter/Transceiver, and Synchronization Signal are presented in Subsections a, b, and c of this section, respectively. The schematic design of the Control and Data Voter is provided in Section 2 of Appendix C.

a) Control Voter

The R3081 has eight outputs that act as control signals in this design: four Byte Enable signals, Burst (BURST*), Read (RD*), Write (WR*), and Data Enable (DATAEN*). When these signals leave their respective processors, they are combined into two 4-bit busses, BECPUx[3..0] and CTRLCPUx[3..0]. The small x in the name represents the specific CPU the signals come from, such as A, B, or C. As an example, CTRLCPUA[0..3] is the BURST*, RD*, WR*, and DATAEN* signals from CPU A bussed together, where BURST* is bit zero, RD* is bit one, WR* is bit two, and DATAEN* is bit three.

The Control Voter design uses two of the VOTER4 macros in parallel. The first macro takes the three CTRLCPUx[3..0] busses and the Force signal as inputs to generate the VOTCTRL[3..0] and CVOTERR[3..0] buses. This is done by performing a majority vote and comparison on the three input busses and passing the appropriate output based on the Force signal, as described in the Address Voter presentation. The second macro generates the VOTBEN[3..0] and BEVOTERR[3..0] busses in the same manner using the three BECPUx[3..0] busses and Force signal. The VOTCTRL and VOTBEN busses are then routed to their respective output pins. The two 4-bit busses of voter error signals are individually ANDed together to form the CVOTERR signal. This signal is combined with the Synchronization Signal, which is discussed later in this section, prior to being routed to its output pin, where it is used by the Memory Control PLD to signal a miss compare.

b) Data Voter/Transceiver

The Data Voter/Transceiver is designed the same as the Address Voter with two exceptions: it uses a different macro to perform the voter function and it is designed to be bidirectional. For the voter function, a new macro named VOTER8 was designed. Although the macro redesign was not required, it was done to help the readability of the top-level design. The VOTER8 macro is designed in the same manner as the VOTER4 macro except eight 3-bit Voter/Error Detectors are used in parallel instead of only four, which gives it an 8-bit wide path instead of the 4-bit wide path of the VOTER4 macro. This reduced the number of macros needed to vote a 32-bit bus to four. The four 8-bit error busses generated by the four VOTER8 macros in the Data Voter/Transceiver are combined in the same manner as the error busses in the Address Voter to form the DVOTERR signal, which is also combined with the Synchronization Signal discussed in the following section. The combined signal is then passed on to its respective output pin.

Because the information passed across the data bus must flow in both directions, the Data Voter/Transceiver had to be designed for bidirectional data flow. The three data busses leaving the processors on write cycles need to be combined through the vote logic and passed on to the Memory System. On reads, the single data bus coming from the Memory System must bypass the vote logic and be split into three copies that are passed back out to the processor's A/D busses. This was done using multiple paths separated by tri-state buffers. The RDDATAEN* and WRDATAEN signals from the Memory Controller PLD are used to control the enable pins of the tri-state buffers. Figure 4.6 shows how the transceiver portion of the Data Voter/Transceiver was designed for each of the thirty-two bits on the data bus.

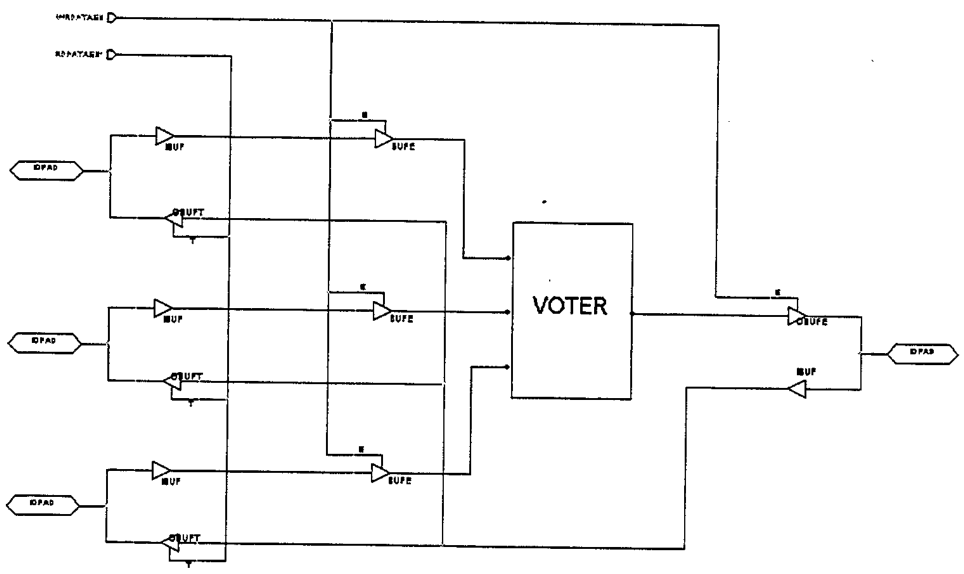


Figure 4.6. Transceiver Logic Design

The Transceiver logic is designed to flow through and around the voter logic. On write bus cycles, data flows from left to right across the diagram, with the three input pins being driven by their respective processors. The Memory Control PLD will assert the WRDATAEN signal, which enables the BUFE tri-state buffers. The data flows through each of the processor input line tri-state buffers, the vote logic, the output tri-state buffer, and to the output pin. The voted data also returns towards the input pins, but is stopped by the disabled BUFT tri-state buffers, which are controlled by the RDDATAEN* signal.

On read bus cycles, data flows from right to left across the diagram, with the single input pin being driven by the Memory System. As discussed earlier, the Memory Control PLD asserts the RDDATAEN* signal and de-asserts the WRDATAEN signal on read cycles. The signal on the input pin is routed through the input buffer and to the three output BUFT tri-state buffers. The asserted RDDATAEN* signal enables the tri-state buffers and lets the data flow through to the processor side pins. The de-asserted WRDATAEN* signal prevents the read data from flowing back into the vote logic.

c) Synchronization Signal (SYNC)

Because skew will exist between the processors, the information on the Address, Control, and Data busses will not arrive at the voters at exactly the same time. Because of the way the Vote Error signal is designed, it will detect these differences and will be asserted whenever the three signals do not match, including when the bus is transitioning and not expected to match. The purpose of the Synchronization signal is to prevent these 'false' voter error assertions from leaving the Voter FPGAs. This was done by ANDing the outputs of the Voter Error signals with the Synchronization signal, which will prevent the error signal from leaving the FPGA unless SYNC is asserted.

The SYNC signal was designed to be asserted during both read and write cycles while valid data was expected to be on the Address, Control, and Data busses. Since the address being used for the current bus cycle is captured by the demultiplexer, the Address bus is held constant once ALE has been de-asserted. As for the Control bus, its value is asserted early in the bus cycle and maintained throughout. The Data bus was the key to the SYNC signal. On reads, RDCEN* is used to indicate valid data is on the bus. On writes, the processor drives the bus shortly after ALE is de-asserted. The ACK* signal is then used to terminate the bus cycle. Since both the RDCEN* and ACK* signals indicate there is valid data on the bus, they were used to generate the SYNC signal, along with the VOTRD* and VOTWR* signals. For the read portion of the SYNC signal, VOTRD* is ANDed with RDCEN*. For the write portion, VOTWR* is ANDed with ACK*. These two signals are then ORed together. Since these signals are active low, the AND-OR is logically an OR-AND, which is shown in Figure 4.7. The combined signal is then inverted and ANDed with an inverted SYSCLK, which is accomplished logically with a NOR gate. The combination with the inverted SYSCLK holds SYNC from being asserted until the second half of the clock cycle, which gives the data bus ample time to settle after the memory system drives the bus on write cycles. The SYNC signal is then routed to its output pin.

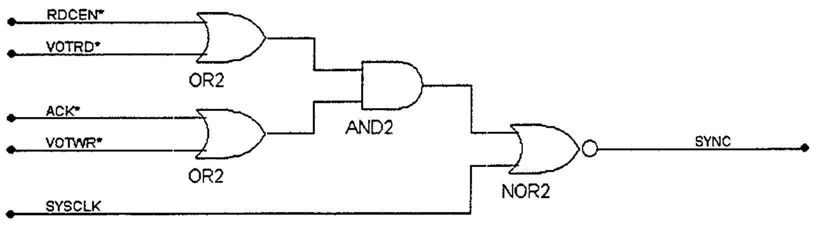


Figure 4.7. SYNC Signal Design

4. System Controller FPGA

As discussed before, LT Payne used the Verilog Design Suite to act as the System Controller in the simulated design that was presented in Ref. [9]. The hardware implementation chose to design the System Controller in an FPGA, since they are one of the few programmable devices that are robust enough to hold all the functionality required of the System Controller. As the design progressed and the test plan started to take shape, the priority assigned to the design of the System Controller began to decrease. Although no testing was conducted during this phase of the project, the initial plan for testing the system is expected to progress as follows:

- Test and analyze system support signals to ensure the system comes out of power-on-reset correctly.
- Test system in single microprocessor mode and validate correct operation of each processor individually.
- Test TMR mode of operation, without collecting bus data.

Because the System Controller is not required to perform the first two sets of tests and the other two Voter FPGAs and the PLDs are, the priority was placed on the design of the other programmable devices. This section presents the initial design of the System Controller, although its implementation was left for follow-on work.

The System Controller serves several functions in the TMR system. It enables the interrupts for the microprocessors after resets, initializes the Control UART, collects and transmits FIFO data, and controls the mode that the TMR R3081 system functions in. The following subsections describe the requirements for each of the System Controller functions and the initial design considerations in implementing them. Since most of the functions are independent and translate easily to finite state machines (FSM), it is recommended that future work use the Xilinx Foundation software to implement the System Controller, since FSM design is one of the design alternatives provided by the software.

a) Interrupt Enable (INTEN)*

The INTEN* signal is used by the Mode Select hardware to multiplex the INT*[3..5] bus of the microprocessors. While the INTEN* signal is not asserted, the 74AHCT541 buffer's outputs are tri-stated and the INT*[3..5] bus is driven by pull-up/pull-down resistors, which are used to select the mode the processors operate in. Since the mode inputs are latched in by the RESET* signal's rising edge, the INTEN* signal should be asserted after RESET* is negated and before the microprocessor's interrupts are enabled.

The only two signals required in the generation of the INTEN* signal are SYSCLK and RESET*. RESET* should be tied to the D input of a flip-flop and SYSCLK to the clock input. The Q output of the flip-flop should then be inverted to generate the negative logic INTEN* signal. This design generates the INTEN* signal on the first rising edge of SYSCLK after RESET* is negated. Since the generation of the

INTEN* signal does not rely on any other function within the System Controller, this function can and should happen simultaneously with the other functions. A diagram of the INTEN* function is provided in Figure 4.8.

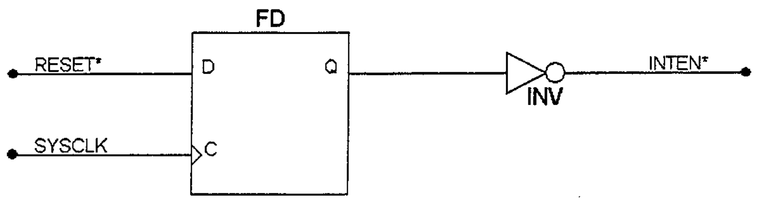


Figure 4.8. INTEN* Signal Design

b) Control UART (CUART) Initialization

Since the majority of the System Controller functions revolve around the collection and transmission of data between the TMR system and the HCI, the CUART plays a very large supporting role to the System Controller. Neither the FIFO Data Transfer nor the TMR Mode Control functions can happen before the CUART is initialized. Therefore, it is very important that this function take place as early in the system initialization process as possible.

The CUART initialization is the process of configuring the UART to communicate with both the CTRLDATA[0..7] bus and the device on the opposite end of the serial port. The minimum initialization requirements for the CUART are to setup the communications channel (baud rate, parity, even/odd, number of start and stop bits) and to select the operation mode (polled or interrupt). If the interrupt mode is selected, the initialization should also enable the interrupts. This is done through access to the

CUART Control Registers, which can be written to and read from. The specific values used in the control registers for the initialization of the UART are developed in Ref. [23].

The System Controller should be able to read from and write to the UART in order to accomplish this function. The state machine should be designed to follow this pattern for a write cycle:

- Assert the CUART chip select signal (CUARTCS*),
- Assert the Control Register address on CTRLADDR[2..4],
- Assert CUARTADS* (From Address Voter),
- Assert data on CTRLDATA[0..7],
- Assert Read Enable signal (CURDEN*),
- Negate CURDEN* (Latches data into register),
- De-assert CTRLDATA[0..7] and CUARTCS*.

The state machine should also be able to produce this pattern of signals for a read cycle:

- Assert the CUART chip select signal (CUARTCS*),
- Assert the Control Register address on CTRLADDR[2..4],
- Assert CUARTADS* (From Address Voter),
- Assert Write Enable signal (CUWREN*),
- Read data off CTRLDATA[0..7],
- Negate CUWREN* and CUARTCS*.

In order to accomplish the read and write operations, all the signals required for the read and write functions were routed between the System Controller FPGA and the CUART.

c) FIFO Data Collection

After the CUART is initialized, the System Controller must collect the FIFO data before it can transfer it to the HCI. The final concept of data collection involves a FIFO Array that continuously collects data. The initialization of the FIFO Array would have the Write Pointer ten addresses ahead of the Read Pointer. The Read and Write Pointers are pointers inside the FIFO that determine which location is read from and written to. While normal execution of instructions occur, the FIFOs perform two read/"dummy write" pairs for each bus cycle. It is referred to as a dummy write because the output enables are not asserted and it only serves to increment the Read Pointer. The first read/"dummy write" collects the address information off the A/D busses and discards the address information from five bus cycles previous. The second read/"dummy write" serves the same function, only for the data portion of the bus cycle. This keeps five bus transactions in the FIFO Array at all times. When a VOTINT* is detected, the "dummy write" is discontinued while the bus information is collected and is restarted after all the FIFO data has been transferred to the HCI. This scheme provides the five bus cycles prior to the error and the internal register contents of the processors to the HCI for analysis.

Because of the large scale of the project, the approach taken with this function of the System Controller was to design a function that would collect the A/D bus information during the interrupt handler. This fulfilled the minimum requirement of providing the contents of all the internal registers of the three processors to the HCI for analysis. Once the system is running and has been tested, this function can easily be expanded to provide the additional five bus cycles.

For the minimum functionality, the System Controller only requires five signals, VOTINT*, SYSCLK, VOTWR*, FWRCLK0, and INTCS*. The System Controller monitors VOTINT* for indication of a voter error. When VOTINT* is asserted, the System Controller uses the write enable strobe (FWRCLK0) of the FIFO to write both the address and data information from the A/D bus into the FIFOs. The

FWRCLK0 signal is generated by combining the SYSCLK and VOTWR* signals. SYSCLK is ANDed with an inverted VOTWR*. Since all the write bus cycles are two clock periods long and the data is latched into the FIFO on the rising edge of the write clock, the first rising edge of FWRCLK0 latches the address and the second rising edge latches the data. Since this signal should not be active at all times it should have an enable that is based on the VOTINT* signal. This can easily be implemented by latching VOTINT* into a D flip-flop and using its inverted output to enable the FWRCLK0 signal. After the interrupt handler has finished storing all the processor registers to memory, it should acknowledge the interrupt, which negates the VOTINT* signal. With the VOTINT negated, the FWRCLK0 signal will be disabled and data collection will cease. A diagram of the FWRCLK0 signal design is shown in Figure 4.9.

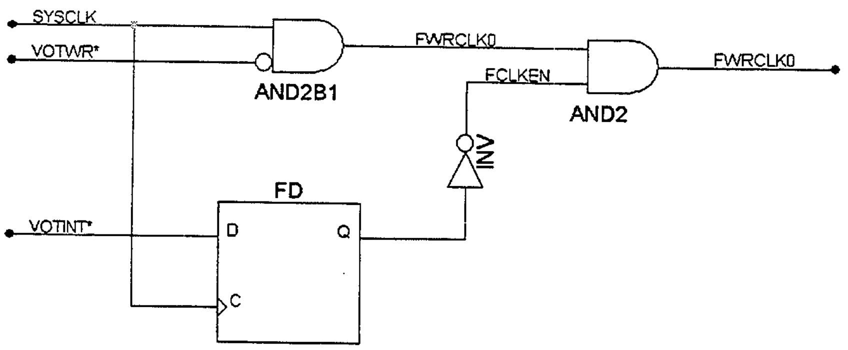


Figure 4.9. FWRCLK0 Signal Design

Since the processors complete the interrupt handler and re-enable their interrupts before the System Controller can transfer all the FIFO data to the HCI, the System Controller must be able to determine if an additional interrupt has occurred while data was being transferred from the previous one. Implementing an up/down counter in the System Controller can do this. The Data Collection system will provide the input for the up counter and the Data Transfer system will provide the input for the down counter. When the INTCS* signal is asserted and negated, the counter will increment by one to indicate a Vote Error Interrupt has been serviced. The fact that the counter contents is not equal to zero is the signal for the Data Transfer system to begin transferring data to the HCI.

d) FIFO Data Transfer

The FIFO Data Transfer system has two functions. First it transfers a Header byte to the HCI to indicate which FIFO's data is about to be transferred. It then transfers the complete contents of that FIFO to the HCI. Because the data transfer is a cyclical operation, the best way to implement it is by using state machines. This section will explain the Header byte and the state machine designs.

(1) Header Byte Design. When the HCI starts receiving data from the TMR system, it has to know what the data represents in order to rebuild the data structure and to conduct analysis on in. This information comes from two locations, the interrupt handler and the Header Byte. By knowing the instructions the interrupt handler is performing and the order it is performing them, the HCI can determine exactly what each byte stored in the FIFOs represent. The list may look something like: R1, R2, ... , R31, PC, Status, Cause...

The Header also provides information regarding the rebuilding of the data structure. The Header contains two elements of information, which processor the data originated from and which FIFO in the array the byte corresponds to. The three processors in the TMR system are referred to as Processor A, B, and C. The fifteen FIFOs in the system are broken down into arrays of five assigned to each processor, which are numbered zero through four. It takes two bits of the Header byte to identify the three processors and three bits to identify the FIFO. The breakdown of the header word is presented in Figure 4.10. Bits 0 and 1 make up the Processor Field, bits 2, 3, and 4 make up the FIFO Field and bits 5, 6, and 7 are reserved for future use.

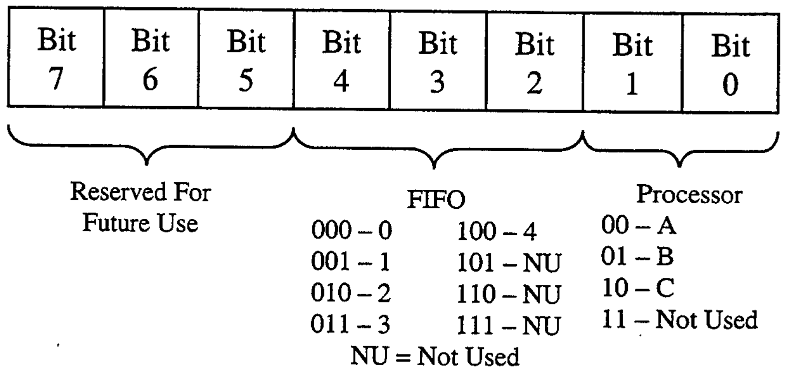


Figure 4.10. Header Byte Design

For each of the three processors, the data from one bus cycle is split into ten bytes with two bytes stored in each FIFO. The first byte corresponds to the address portion of the bus cycle and the second byte corresponds to the data portion. When the Data Transfer system transfers the data to the HCI, it will transfer all of the data corresponding to the current vote error from one FIFO before moving on to the next FIFO. This method of transferring the data was preferred because it reduces the latency of the transfer by not having to switch between FIFOs after each byte. The state machines of the FIFO Data Transfer system, which are explained next, coordinate the movement of the data from the FIFOs to the UART for transmission to the HCI.

(2) State Machine Design. Because the majority of the elements of the FIFO Data Transfer system involve sequential tasks, the system is best suited to being designed as a set of finite state machines (FSMs). The FIFO Data Transfer system is designed as a hierarchical set of five FSMs: Data Transfer FSM, Processor Select FSM, FIFO Select FSM, Byte Select FSM, and Data Write FSM. This section presents their design and discusses their interactions.

The Data Transfer FSM is the uppermost FSM in the system. It monitors the Interrupt Counter discussed in the FIFO Data Collection section. When the counter is no longer equal to zero, the Data Transfer FSM changes from the Wait state to the Transfer Data state. While in this state, the FSM asserts an enable for the Processor Select FSM and waits for the Transfer Complete signal to be returned, which signals the completion of the FIFO data transfer. When the Transfer Data operation is completed, the FSM returns to the Wait state and checks the value of the Interrupt Counter, which the Processor Select FSM decrements. If the counter is zero, the FSM will idle in the Wait state. If the counter is still not equal to zero, the FSM returns to the Transfer Data state. The Data Transfer FSM is given in Figure 4.11.

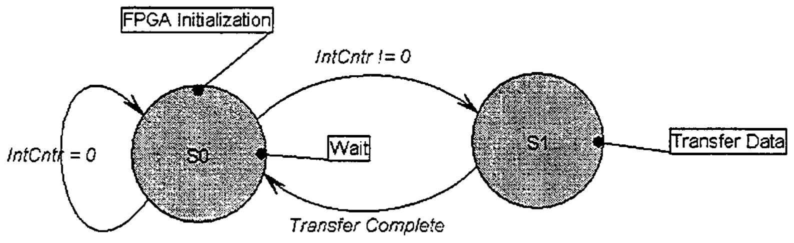


Figure 4.11. Data Transfer FSM Design

The next FSM is the Processor Select FSM. The purpose of this FSM is to sequence through the three processors' FIFO arrays during the data transfer. This FSM waits in its initial state until enabled by the Data Transfer FSM. When enabled, the FSM transitions to the Processor A state, where it asserts an enable for the FIFO Select FSM and waits for a CPU Complete signal. The CPU Complete signal indicates that the FIFO contents of the current processor have been transferred to the HCI. The FSM then completes this operation for Processors B and C. After Processor C's FIFO contents are transferred, the next state decrements the Interrupt Counter and asserts the Transfer Complete signal. The FSM then transitions back to its Wait state. The Processor Select FSM is shown in Figure 4.12.

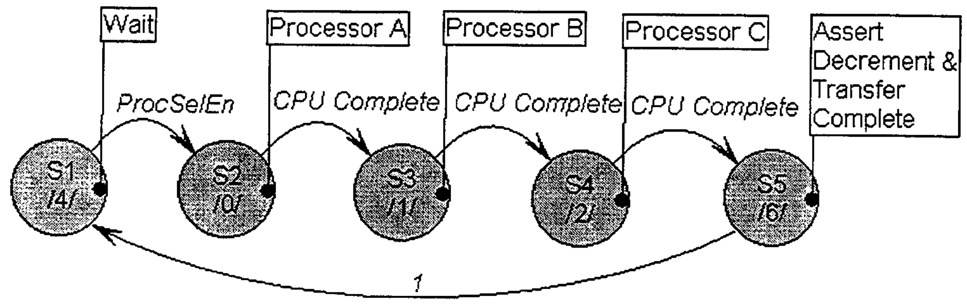


Figure 4.12. Processor Select FSM

The next FSM in the hierarchy is the FIFO Select FSM. The purpose of this FSM is to sequence through each of the five FIFOs in the FIFO array during a data transfer. This FSM waits in its initial state until enabled by the Processor Select FSM. When enabled, the FSM transitions to the FIFO 0 state, which corresponds to the most significant byte (MSB) of the A/D FIFOs. During the FIFO 0 state, the FSM enables the Write Data state machine and waits for a FIFO Complete signal. The FIFO Complete signal indicates that the Write Data FSM has transferred both the Header and the contents of the current FIFO to the UART. When the FIFO Complete signal is received, the FSM transitions to the next state and the process is repeated for each FIFO in the array. The sequence for the FIFOs is A/D MSB, A/D next to MSB, A/D next to LSB, A/D LSB, and Control byte. After the Control FIFO has been transferred, the next state in the FSM asserts the CPU Complete signal and then transitions to its Wait state. The FIFO Select FSM is shown in Figure 4.13.

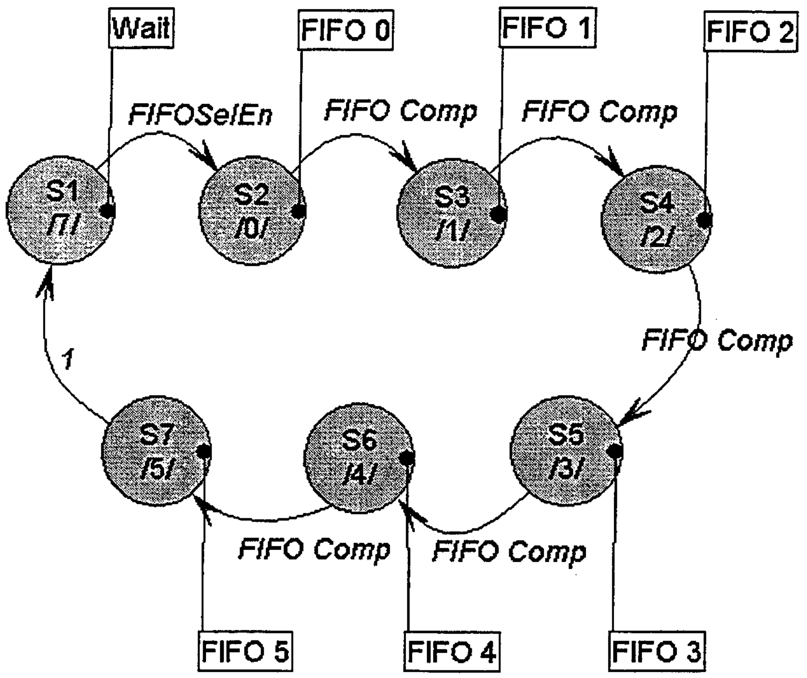


Figure 4.13. FIFO Select FSM Design

The Byte Select FSM is the next FSM in the hierarchy. The purpose of this FSM is to sequence through the Header and each of the bytes of data in the FIFO during a data transfer. This issue was complicated by the fact that the code to the O/S was not available. In order to build a Rapid Prototype of the HCI that could be tested, the initial data block agreed upon between the hardware and software system designers was 41 of the processor's internal registers. The 41-register block contains the 10 CP0 control registers and 31 of the 32 general-purpose integer registers. R0 was not transferred because it is a constant zero. Since the bus is sampled twice for each write cycle, the System Controller is expected to put 82 bytes of data in the FIFO during the interrupt handler. When the Header is added to this, the Byte Select FSM has 83 bytes of data to sequence through and Wait and FIFO Complete states. This brings the initial total to 85 states for this FSM. It waits in its initial state until enabled by the FIFO Select FSM. When enabled, the FSM transitions to the Header state, which asserts the Data Write Enable and waits for the Byte Complete signal. The Byte Complete signal indicates that the Write Data FSM has transferred the Header byte to the UART. When the Byte Complete signal is received, the FSM transitions to the next state and the process is repeated for each byte in the FIFO. After the last byte in the FIFO has been transferred, the next state in the FSM asserts the FIFO Complete signal and then transitions to its Wait state. Since the Byte Select FSM is so large, only the initial portion of the design is shown Figure 4.14.

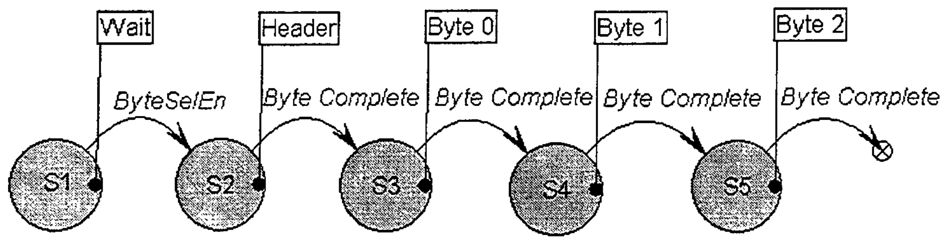


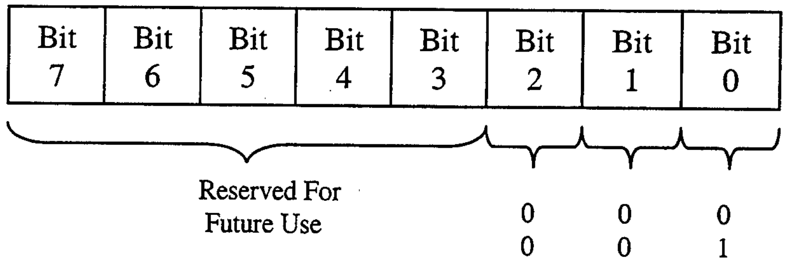
Figure 4.14. Byte Select FSM Design

Finally, the last FSM in the FIFO Data Transfer function is the Write Data FSM. This state machine is responsible for issuing the control signals required to transfer data from the FPGA or FIFO to UART. This FSM was described in the CUART Initialization section. After the selected byte is latched into the UART, the Write Data FSM asserts the Byte Complete signal and returns to its Wait state, which wraps up the design of the FIFO Data Transfer system.

e) TMR Mode Control

The last functionality to present in the System Controller is the TMR Mode Control. The TMR Mode Control serves two purposes, it allows the user to remotely reset the TMR system and to select between the TMR and Single Processor Modes of operation. This section presents the design of the TMR Mode Control. Although the TMR Mode Control performs two functions, it can be implemented in a single 8-bit register with the outputs tied to the signals required to initiate the required operations. Bit 0 is used to select between the TMR and Single Processor Mode. Its output is the FORCE signal. When this bit is a zero, the system operates in the TMR Mode. When it is a one, the system operates in the Single Processor Mode. Bit 1 initiates a system-level reset by routing the inverted output of the register to the

SYS_RESET output of the FPGA. Bit 2 initiates a board-level reset by routing the inverted output of the register to the BRD_RESET output of the FPGA. A diagram of the TMR Mode Select Register is given in Figure 4.15.



Bit 2	Bit 1	Bit 0	
0	0	0	TMR Mode
0	0	1	Single Processor
0	1	X	System Reset
1	X	X	Board Reset

Figure 4.15. TMR Mode Select Register Design

C. DETAILED SYSTEM TIMING ANALYSIS

With all the hardware specified and the programmable devices designed, the next step in the process is to conduct a detailed timing analysis of the system. The purpose of the timing analysis is to determine if the system operates in the manner expected and to ensure that no bus contentions exist. A bus contention occurs when multiple devices are driving the same lines at the same time, which could lead to a destructive fault.

Since the processors can conduct three different types of bus transactions, the timing analysis was performed on each type of bus transaction and is presented in the same fashion: Single Word Read Cycles, Quad Word Read Cycles, and Single Word Write Cycles. Additionally, each device that the processors can interact with for a particular bus cycle had to be analyzed. The following sections present the results of the timing analysis. A timing diagram has been generated using Timing Designer

Professional, a software tool developed by Chronology. Like transactions have been combined where possible.

1. Single Word Read Cycles

Single word reads can be characterized by three phases: the processor initiates the bus cycle, the Memory System responds to the request, and the processor terminates the bus cycle. When the timing analysis was conducted for all the devices, it was found that the SRAM, UART, Timer, and Interrupt all have the same timing diagram. Only the EPROM had a different timing analysis for single word read cycles.

The processors initiate the bus cycle by asserting the appropriate control signals on a rising edge of SYSCLK*. In this case, the RD* signal is asserted, A/D bus is driven with the address, and ALE is asserted. BURST* is not asserted for single word reads. The assertion of ALE causes the latches in the A/D demultiplexer to become transparent and allow the address information to continue to the Address Voter FPGA. Near the next falling edge of SYSCLK*, ALE is negated and the processors turn the A/D busses to receive the data supplied by the Memory System. The negation of ALE latches the address into the transparent latches. The processors will wait in this state for the data indefinitely, unless the BUSERR* signal is asserted.

The address busses are voted and decoded by the Address Voter FPGA and passed on to the Memory Systems address lines and chip selects. The Control Voter also votes all the Control lines during this time, which supplies the signals required by the Memory System to respond to the bus cycle. The assertion of VOTRD*, without VOTBURST*, signals the Memory Control PLD that a single word read cycle has been initiated.

Regardless of which chip select signal is asserted, the Memory Control PLD asserts the RDEN* and RDDATAEN* signals. This enables the output drivers of the device being read from and directs the data to flow from the Memory System towards the processors in the bus transceivers and Data Voter FPGA. The asserted chip select signal determines how many wait states are generated by the Memory Control PLD. The EPROM was the only device that required a single wait state, all the other devices could conduct zero wait state reads. For the zero wait state devices, RDCEN* was asserted at the same time as the RDEN* signal and one clock cycle later for the EPROM, which signaled the processors that the Memory System had finished responding to the bus cycle.

The assertion of RDCEN* by the Memory Controller PLD must occur prior to the rising edge of SYSCLK*. This requirement is easily attained because SYSCLK is used to drive the PLDs state changes and it is very nearly exactly out of phase with SYSCLK* due to the very small propagation delay of the inverting buffer. The processors then latch the data off the A/D buses on the next falling edge of SYSCLK*, along with negating all the control signals asserted for the bus cycle. The zero wait state read takes two SYSCLK* cycles to complete and the single wait state read takes three clock cycles to complete. The timing diagrams for the zero and single wait state timing analysis are presented in Figures 4.16 and 4.17, repectively. The diagram indicate that no bus contentions exist on any of the busses.

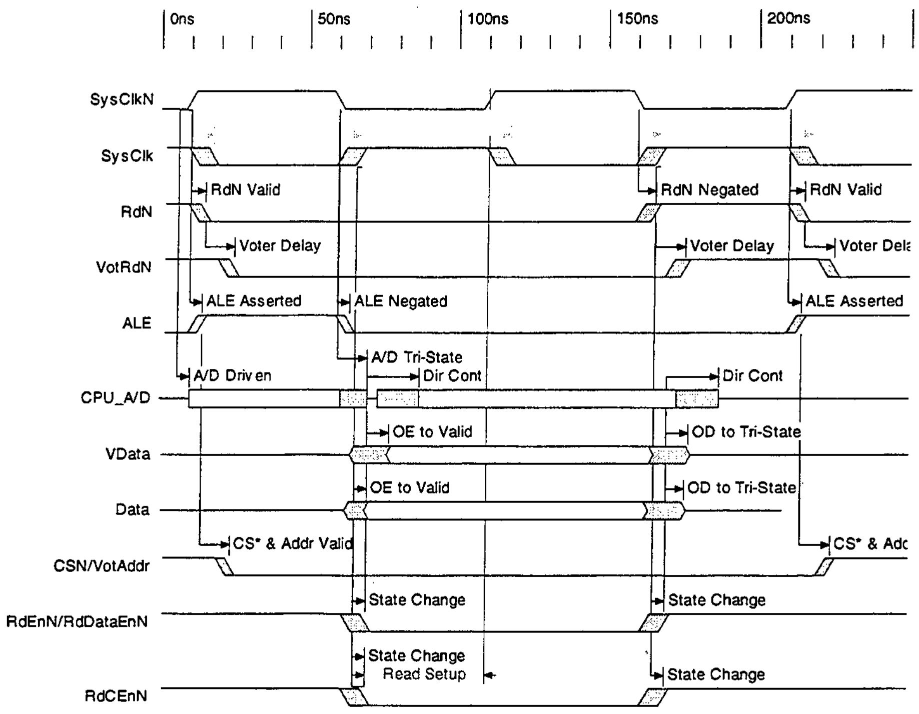


Figure 4.16. Zero Wait State Read Timing Diagram

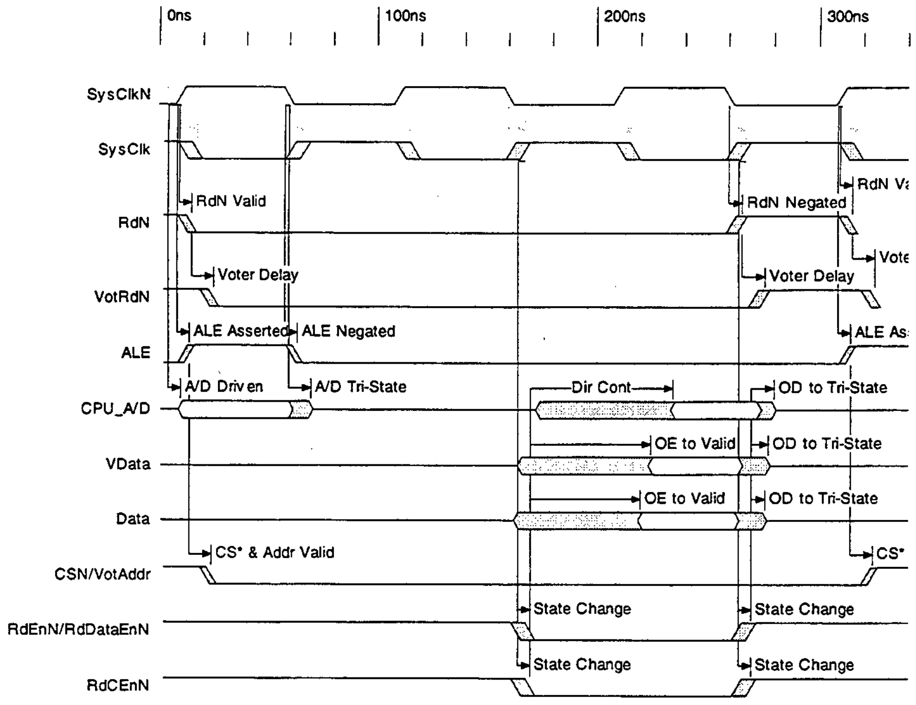


Figure 4.17. Single Wait State Read Timing Diagram

2. Quad-Word Read Cycles

In order to reduce the latency involved in setting up read operations, the R3081 supports quad-word reads. The bus cycle is initiated in the same fashion as a single word read, except the processors also assert the BURST* signal. When the Memory System responds with the RDCEN* strobe, the processors latch the data off the bus and increment the CPUADDR[3..2] bus from 00 to 01. The Memory System then responds with another RDCEN* strobe and the process is repeated four times, which reads the addresses corresponding to 00, 01, 10, and 11 on CPUADDR[3..2]. After the fourth RDCEN*, the processors will terminate the read cycle. In order to provide better response, the R3081 lets the designer apply the ACK* signal up to four clocks before the last data word is latched in. This lets the processor core start processing the words that have already been latched into the read buffer, as long as the last word is in the buffer by the time the processor requires it.

When the timing analysis was conducted, it was determined that the only two devices capable of performing quaded reads are the SRAM and EPROM. The other devices do not have linear memory spaces or multiple registers that require successive writes. When the SRAM's and EPROM's timing diagrams were compared, they were very similar. As expected, the SRAM was able to conduct zero wait state reads for each of the four reads. What was unexpected, was that the EPROM was able to conduct zero wait state reads for the last three reads after an initial single wait state latency during the first read operation. Because the SRAM can conduct four single-word reads or a single quad-word read in eight cycles, there is no savings in doing this. The EPROM is able to reduce the number of cycles required to read four words during the quad-word read due to the added delay of waiting for the Memory Control PLD to toggle the RDCEN* signal. This adds a complete clock cycle while the RDCEN* is taken high for the EPROM address to be accessed. The EPROM quad-word read saves a total of three clock cycles over four single-word reads. Because the timing diagrams for the quad-word reads were so long, they were split into two parts for legibility. The timing diagram for the SRAM Zero Wait State Quaded Word Read is given in Figures 4.18 and 4.19. The timing diagram for the EPROM Single Wait State Quaded Word Read is given in Figures 4.20 and 4.21. As with the single-word read cycles, the quad-word read timing diagrams indicate that no bus contentions exist.

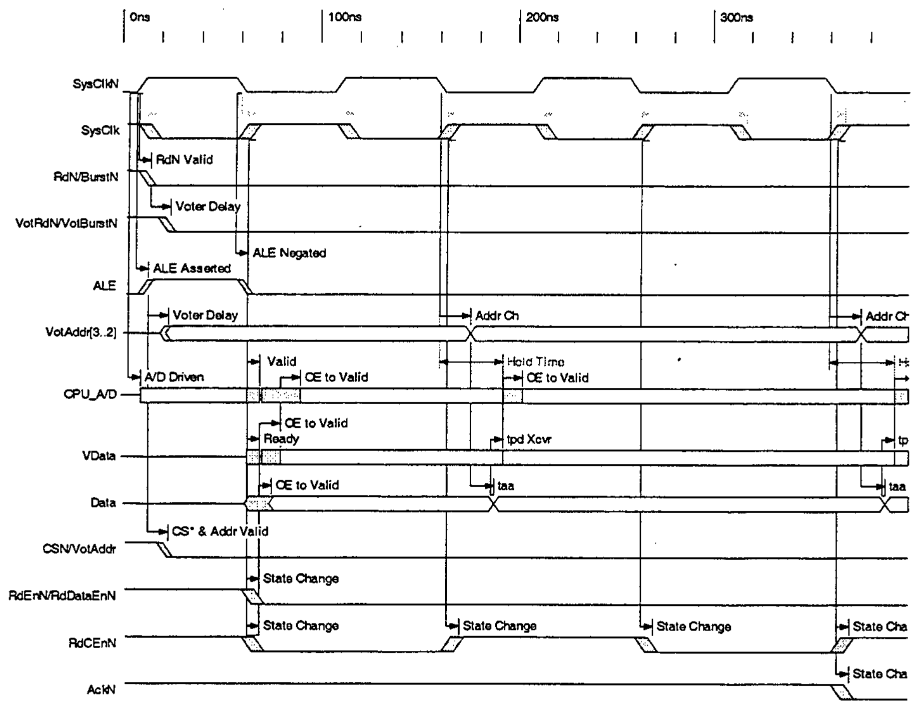


Figure 4.18. Zero Wait State Quaded Word Read Timing Diagram (1st Half)

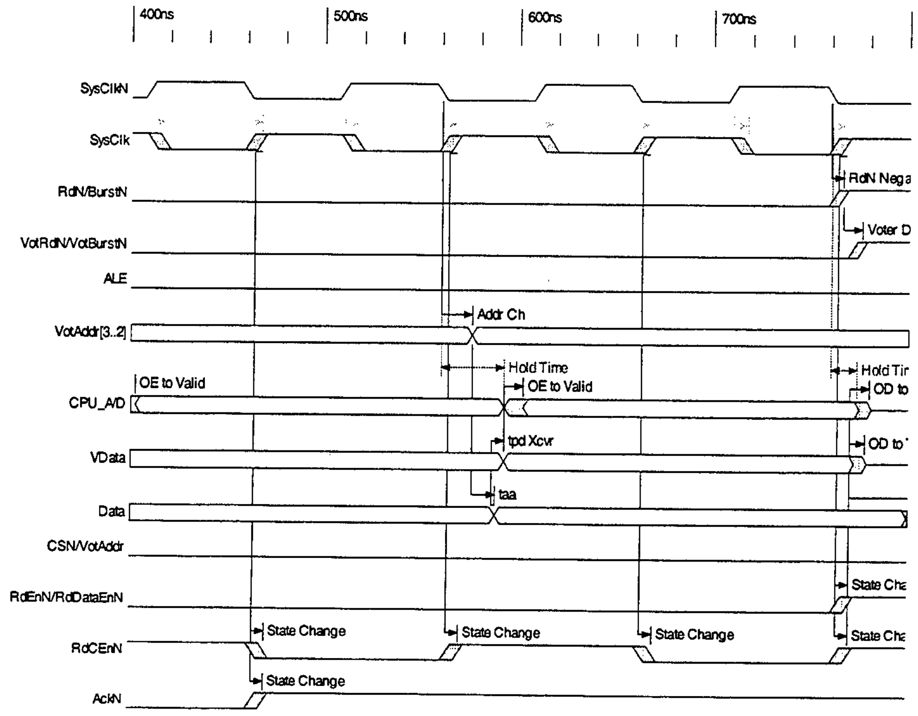


Figure 4.19. Zero Wait State Quaded Word Read Timing Diagram (2nd Half)

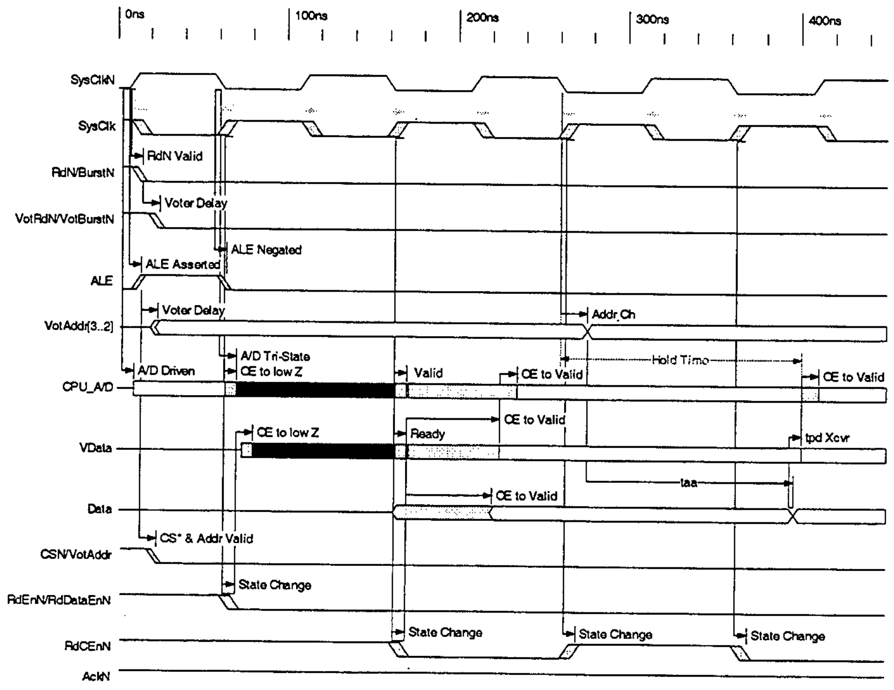


Figure 4.20. Single Wait State Quaded Word Read Timing Diagram (1st Half)

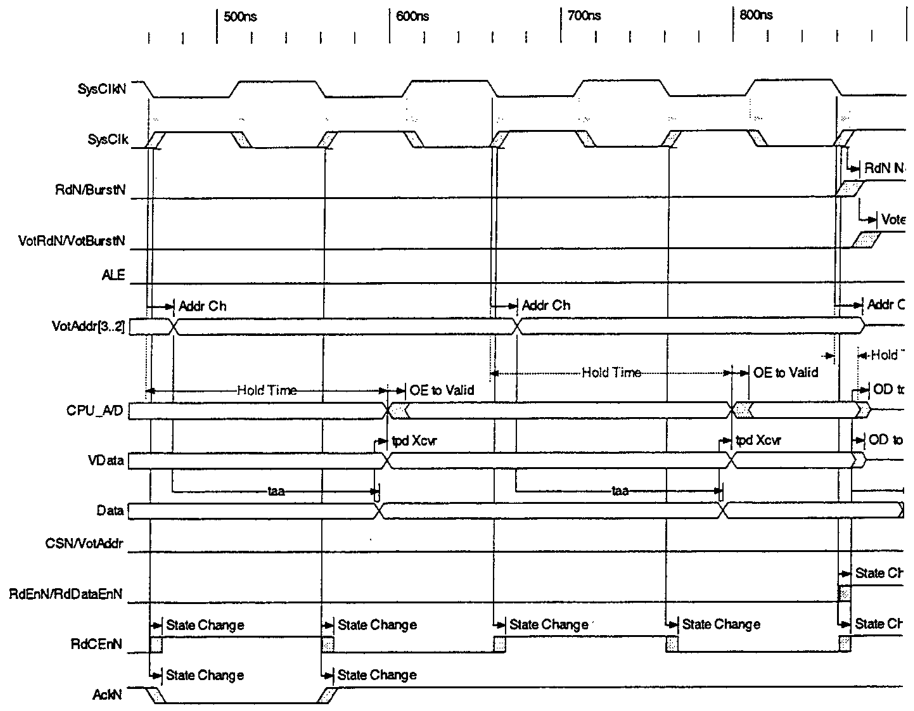


Figure 4.21. Single Wait State Quaded Word Read Timing Diagram (2nd Half)

3. Write Cycles

The R3081 also conducts write cycles in three phases, which are very similar to the phases of the read cycle. First, the processors initiate the bus cycle by asserting the WR* control line, ALE, and the A/D bus with the address. After the processors negate ALE, they drive the busses with data instead of waiting for the Memory System to respond. After the processors have initiated the bus cycle, the Memory Controller responds to the VOTWR* signal by asserting the WRDATAEN and ACK* signals. The WRDATAEN signal sets the transceivers and Data Voter FPGA up to allow data to flow from the processors to the Memory System. The ACK* signal is the signal the processors are waiting for that indicates the Memory System has had sufficient time to process the bus transaction. The processors then terminate the bus cycle.

Because EPROM is not written to, no timing analysis was required for this type of bus cycle. The remaining devices were all able to perform zero wait state writes. Their timing diagram is provided in Figure 4.22. As with the other timing diagrams, there are no bus contentions during the write cycle either.

Although the detailed timing analysis brings together the hardware and software designs and checks them for errors, nothing substitutes for performing tests on the actual circuit boards to determine if they were designed correctly. The next chapter presents the steps taken to turn the implementation design into an actual circuit board with integrated circuits, switches, capacitors, and connectors.

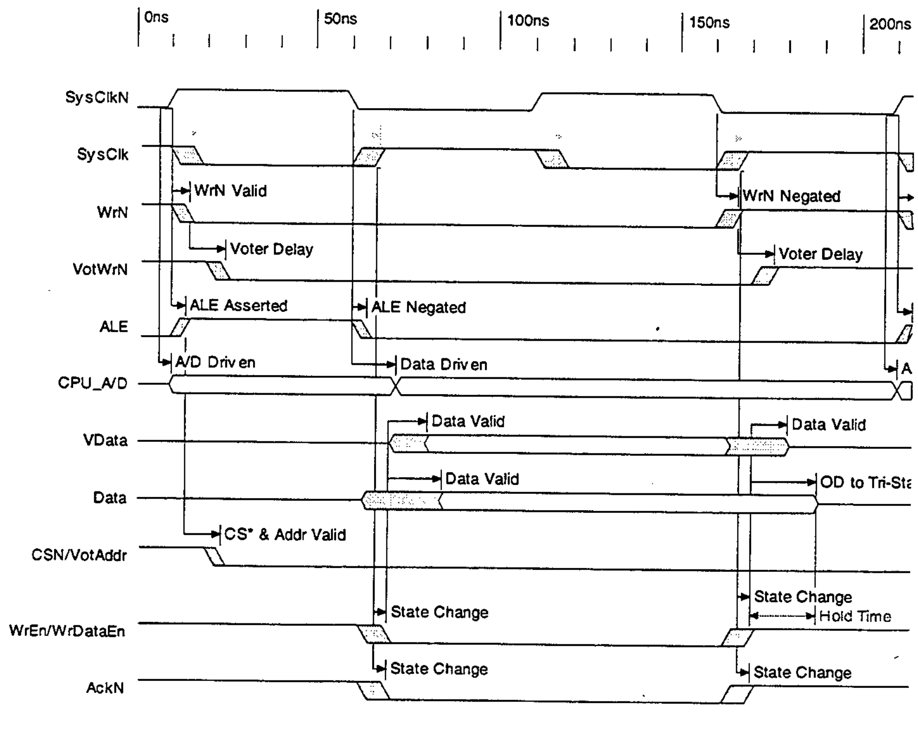


Figure 4.22. Zero Wait State Write Timing Diagram

V. MANUFACTURING AND DESIGN REVIEW

Chapters II through IV presented a simulated design of the TMR R3081 system and the hardware and programmable logic designs required in the implementation of the design. This chapter presents what was done with those designs in order to produce a printed circuit board (PCB) for use in follow-on research and testing.

A. PCB FABRICATION

One of the issues not discussed in the earlier chapters was the selection of a software suite that could be used to draw the TMR system schematics, which ties in with the PCB fabrication. The initial inclination was to use the Verilog Design Suite, since it had been used for the simulated design. Unfortunately, the system had been recently upgraded and all the bugs had not been worked out. After spending the better part of six weeks trying to get the Verilog solution to work, it became apparent that it was not going to be a viable option. When the search turned to PC based software, two schematic capture software tools emerged as the front runners, OrCAD and Protel. After using both of the software's demonstration programs and talking to the contractor that was going to perform the PCB layout, the OrCAD software was chosen over the Protel. The two main reasons the OrCAD software was selected was that it offered better parts libraries and it is the same software that the contractor uses. Since the libraries could be checked prior to starting the design, all the parts required in the TMR system were available in the OrCAD software. Additionally, since the contractor also uses OrCAD, the designs could be imported directly into the PCB layout portion of the OrCAD suite, which also saved time in the manufacturing process.

Along with the OrCAD schematics, the contractor was given a list of requirements concerning the layout of the system. There could be no traces under the processors, the CPUCLK traces should be as close to the same length as possible, and the IC chips used on the PCB should be socketed. The restriction on the traces under the processors comes from Heavy Ion Testing. The traces could interfere with the test by generating additional ions when impacted by the test beam. The new particles could then cause errors in the processor, which would skew the test data. The CPUCLK traces need to be as close to the same length as possible to reduce the skew between the signals. This stipulation was included because the synchronization of the three processors was thought to be the most critical aspect of the design. If there was too much skew between the signals, the processors might not finish a bus cycle without a miss compare. Finally, the socketed parts allow for quick and simple part exchanges for the programmable devices or for upgrades to the other devices.

Once the OrCAD schematics were delivered to the contractor, he created a Net List out of the project. The Net List was then used to create the PCB layout. Before sending the board out to the fabricators, the contractor attempted to order the parts specified in the schematics. It was during this time that the problem with the delivery time on many of the parts selected in the design was discovered. The design was then updated to reflect the changes that needed to be made because of the lack of part availability and the layout was sent to the fabricators. When it returned, the contractor soldered in all the sockets and the parts that were not put in sockets prior to returning the finished board. Figure 5.1 shows a picture of the TMR R3081 PCB with the functional blocks identified.

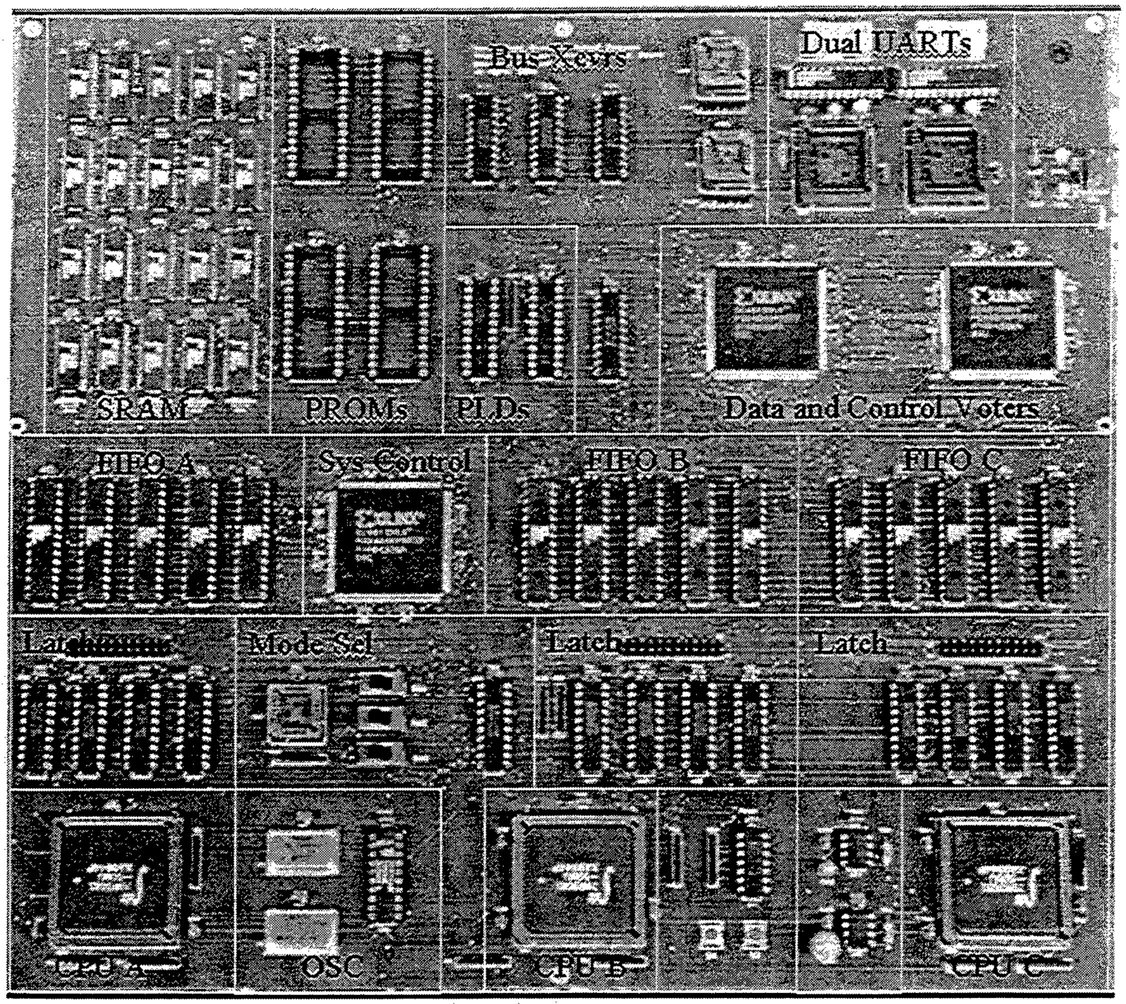


Figure 5.1. TMR R3081 PCB

B. WHITE WIRES

Although the hardware design was well thought out prior to sending it to the contractor for manufacturing, the PLD and FPGA designs had not been completed. This was done to streamline the design process in order to have a working prototype as early as possible. Having the early prototype would allow the software development team time to integrate the O/S and HCI into the design. Unfortunately, the plan did not work as expected and the design of the PLDs and FPGAs revealed additional routings that needed to be made, which are called White Wires. This section presents the design changes that were discovered during the PLD and FPGA designs, which are presented in Table 5.1 for ease of reference.

Signal Name	From	To	Change
TIMERCS*	Address Voter Pin 184	Memory Controller Pin 4	White Wire
RESET*	Memory Enable Pin 23	Memory Controller Pin 13	White Wire
RESET*	Memory Enable Pin 23	Memory Controller Pin 14	Cut
RDDATAEN*	Memory Enable Pin 14	Data and Control Voter Pin 185	White Wire
WRDATAEN	Memory Enable Pin 15	Data and Control Voter Pin 186	White Wire
SYSCLK	Buffer/Driver Pin 11	Data and Control Voter Pin 171	White Wire
RDCEN*	Memory Controller Pin 22	Data and Control Voter Pin 172	White Wire
ACK*	Memory Controller Pin 21	Data and Control Voter Pin 173	White Wire
RESET*	Memory Enable Pin 23	System Controller Pin 144	White Wire

Table 5.1. Table of White Wires and Cuts

When the Timer system was redesigned because the parts were no longer manufactured, the TIMERCS* signal was removed from the design. It was discovered during the Memory Controller PLD design that the “dummy writes” to the Timer required a chip select signal to be generated so the Memory System would generate the proper signals to acknowledge the write, even though nothing was actually being written to the timer.

Another issue discovered during the Memory Controller PLD design required the moving of the RESET* signal. The original PLD design used pin 13 of the PLD as part of the Wait State Counter. Since pin 13 is an input only pin, it could not be used for the feedback options that were required of the counter. The easiest way to remedy this problem was to move the RESET* signal input from pin 14 to pin 13 of the Memory Controller PLD and cut the trace leading to pin 14.

During the detailed timing analysis, a bus contention was discovered on the processors' A/D busses. The VOTRD* and VOTWR* signals were being used to control the direction of data flow through the Data Voter/Transceiver. Because they were driven so early in the bus cycle, the Data Voter/Transceiver would drive the A/D buses while the processors were still driving them with the address values. In order to prevent the bus contention, the RDDATAEN* and WRDATAEN signals were supplied to the Data Voter/Transceiver to perform the direction control function. Since these signals are synchronized to SYSCLK, they provide an additional half cycle of delay before the busses are driven, which prevents the bus contention.

During the design of the SYNC signal in the Data and Control Voter, it was discovered that the control signals supplied to the FPGA were not adequate to generate the SYNC signal. When the timing requirements presented in Chapter IV were analyzed, it was determined that the SYSCLK, RDCEN*, and ACK* signals were required by the Data and Control Voter FPGA to generate the SYNC signal.

The last White Wire needed was discovered during the design of the System Controller FPGA, which was originally going to use the SYS_RESET* signal to signal the assertion of the INTEN* signal. Because the processors are using the synchronous RESET* signal for their reset function, this signal also had to be added to the System Controller FPGA for the INTEN* signal. If the SYS_RESET* signal was used, the INTEN* signal would be asserted too early, which could cause the processors to initialize in the wrong mode.

Once the White Wires and cuts are added to the TMR R3081 system, it will be ready to start the process of testing. The design of the PLDs, FPGAs, and the detailed timing analysis were conducted in conjunction with LT Damen Hofheinz, USN, who will be conducting the testing and follow on research. This was done to provide as seamless a transition as possible. The following chapter concludes this portion of the project and discusses possible areas for follow on research.

VI. CONCLUSIONS AND FOLLOW-ON RESEARCH

The previous chapters have introduced the concept, provided background material, and presented the work completed on this project to date. This chapter will present the conclusions drawn from the project and possible areas for follow-on research.

A. CONCLUSIONS

The high prices and declining availability of radiation hardened devices has caused system designers to turn to using COTS devices in radiation environments. Although the COTS devices offer reduced design-to-orbit time, lower cost, and use of cutting edge technology in the design, they are also much more susceptible to SEUs.

The TMR system implemented during this research provides a tool to reduce the risk posed by the SEUs in two different ways: by testing software algorithms or by masking out the fault. First, the TMR Software Testbed allows testing of fault-tolerant software in a radiation environment without having the expense of placing a test case in orbit. This allows the reactions of the software to be monitored when an SEU is encountered. When the TMR system is used as a hybrid fault-tolerant computer system, the hardware will mask out the error, reset the faulty processor to match the state of the other two processors, and continue processing from the point the error occurred.

During this portion of the project, the simulated design model was expanded, upgraded, and implemented. The system was expanded by the addition of the support elements that were not required in the simulation phase. The addition of the System Controller to the design and its requirement to handle the data transfers between the HCI and the TMR System improve the processors availability for user programs. During the implementation, the integrated circuit chips in the system, with exception of the microprocessor, were selected, the schematic designs were drawn, and the programmable logic devices were designed. Finally, a detailed timing analysis of the complete system design was performed, during which it was determined that there were no bus contentions and all the devices' setup and hold times were met during all modes of operation.

The simulated model of this system from Ref [9] and the hardware implementation presented in this research both prove the TMR concept to be a valid method of dealing with the effects of SEUs on COTS devices. This research advanced the project by implementing the system in hardware, which required the design of the system support elements, I/O space, memory space, FIFO interface, and the System Controller. Although the System Controller implementation was not completed and no developmental testing was conducted, the detailed timing analysis presented in Chapter IV supports the claim that the design will work as intended.

B. FOLLOW-ON RESEARCH

Because of the relatively short time students get to actually conduct research, projects as large as this one are not normally completed by one student. The projects are segmented and each student in the progression will advance the project. This section explains some of the areas where follow-on research can be conducted, such as completion of the TMR Testbed, Radiation Testing, conversion to a space flight board, or applying the TMR design to a state-of-the-art (SOTA) processor.

1. Completion of TMR Implementation

This thesis presented a partially functional TMR R3081 system. The hardware implementation, along with the programming for the Memory Controller PLDs and the Voter FPGAs allow the system to function as a stand alone computer system with the added function of detecting and correcting single bit errors occurring in any of the processors. In order to provide the full capability of the system, the System Controller FPGA needs to be programmed and operationally tested. This will allow the system to store and transmit data collected from each of the processors during a Voter Interrupt to the HCI for analysis. This will also complete the hardware design phase for this system.

Once the hardware is completed, the system will be ready for the software integration, which consists of two parts, the O/S and the HCI, which are detailed in Ref. [12]. The VxWorks O/S is installed on the hardware by loading it onto the EPROMs. It then needs to be tested to ensure the device drivers and the interrupt handlers are configured correctly. The HCI is incorporated into the overall system by connecting its Data and Control Ports to the Data and Control Ports of the TMR system, since they are both stand-alone systems that communicate via two serial cables. The HCI and hardware need to be tested to ensure the communication paths are operating correctly and the data format being transmitted by the hardware matches the format expected by the HCI. When these items are completed, the system will be a fully functional TMR system that is able to detect and correct single errors in any of the processors and provide the data corresponding to that error to the HCI for analysis. The system is then ready for radiation testing and modifications for other uses.

2. Radiation Testing

When the hardware and software integration is completed, the TMR R3081 system will be ready for its operational testing. Because this system is designed to detect and correct radiation induced SEUs, its final operational testing must be done in the presence of radiation. Regional test facilities include the particle accelerators at the University of California, Davis and University of California, Berkeley. These facilities generate beams of high-energy particles that radiate devices placed in their path. By placing one of the processors in the radiation beam, SEUs can be induced in the device and the reaction of the system can be captured and analyzed by the HCI.

The data captured by the HCI during the radiation testing can be used for two different purposes. First, it will validate the concept of a hybrid TMR fault tolerant system. Previous versions of TMR processor systems rebooted the system when an SEU was detected in one of the processors. This system uses the two valid processors, the Voters, and the memory system to reconfigure the faulty processor to match the two valid processors. This technique saves all the time and data that is lost when the recovery plan from an SEU is a complete reboot of the system.

Since the TMR system can be operated in the TMR mode or single processor mode, a test can be conducted to compare the efficiency of TMR hardware fault tolerance to software fault tolerance. One possible test could be to have the system count to a specified value while being subjected to the radiation beam. One test would use fault tolerant software in the single processor mode, while the other test would use software without fault tolerance in the TMR mode. The time that was required to complete the two operations could then be compared.

3. Conversion to Space Flight Board

Once the system has been integrated and tested, the next logical step in its evolution would be to convert it into a space flight board. The new board could then be used to conduct further testing or as a computing node on the satellite. Some of the main topics that would need to be investigated for the conversion are listed below.

- Determination of parts to be replaced with commercially available radhard models.
- Addition of an error detecting and correcting memory space.
- Addition of circuitry to detect SELs and reset the system.
- Weight and power analysis.
- Detailed timing analysis due to changes in system timing.

4. Application to a State-of-the-Art (SOTA) Processor

Although the R3081 microprocessor provides sufficient computational power to analyze fault tolerant techniques, its performance is far behind SOTA processors available today. One possible topic for future research on this project would be to basically start fresh with one of today's SOTA processors. One of the limiting factors of TMR designs in the past has been the additional delay placed in the communication paths by the logic needed to perform the voting. Because the processor cores have gotten so much faster than the memory devices that support them, the additional delay of the voters should be lost in the noise, but that will be left for someone else to prove.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. TMR IMPLEMENTATION ORCAD SCHEMATICS

This appendix contains the schematic for the TMR R3081 System. Figure A.1 is the top-level schematic and each block contained in it refers to one or more other figures. Table A.1 lists the figures in this appendix and the page they appear on.

Figure Number and Description	Page Number
Figure A.1. TMR R3081 Top Level Schematic	116
Figure A.2. System Support Elements Schematic	117
Figure A.3. CPU A Microprocessor and Latch Schematic	118
Figure A.4. CPU B Microprocessor and Latch Schematic	119
Figure A.5. CPU C Microprocessor and Latch Schematic	120
Figure A.6. Address Voter FPGA Schematic	121
Figure A.7. Data and Control Voter FPGA Schematic	122
Figure A.8. PROM Array Schematic	123
Figure A.9. KRAM Array Schematic	124
Figure A.10. URAM0 Array Schematic	125
Figure A.11. URAM1 Array Schematic	126
Figure A.12. URAM2 Array Schematic	127
Figure A.13. URAM3 Array Schematic	128
Figure A.14. Data UART Schematic	129
Figure A.15. CPU A FIFO Array Schematic	130
Figure A.16. CPU B FIFO Array Schematic	131
Figure A.17. CPU C FIFO Array Schematic	132
Figure A.18. System Control FPGA Schematic	133
Figure A.19. Control UART Schematic	134

Table A.1. TMR R3081 System Schematics By Page Number

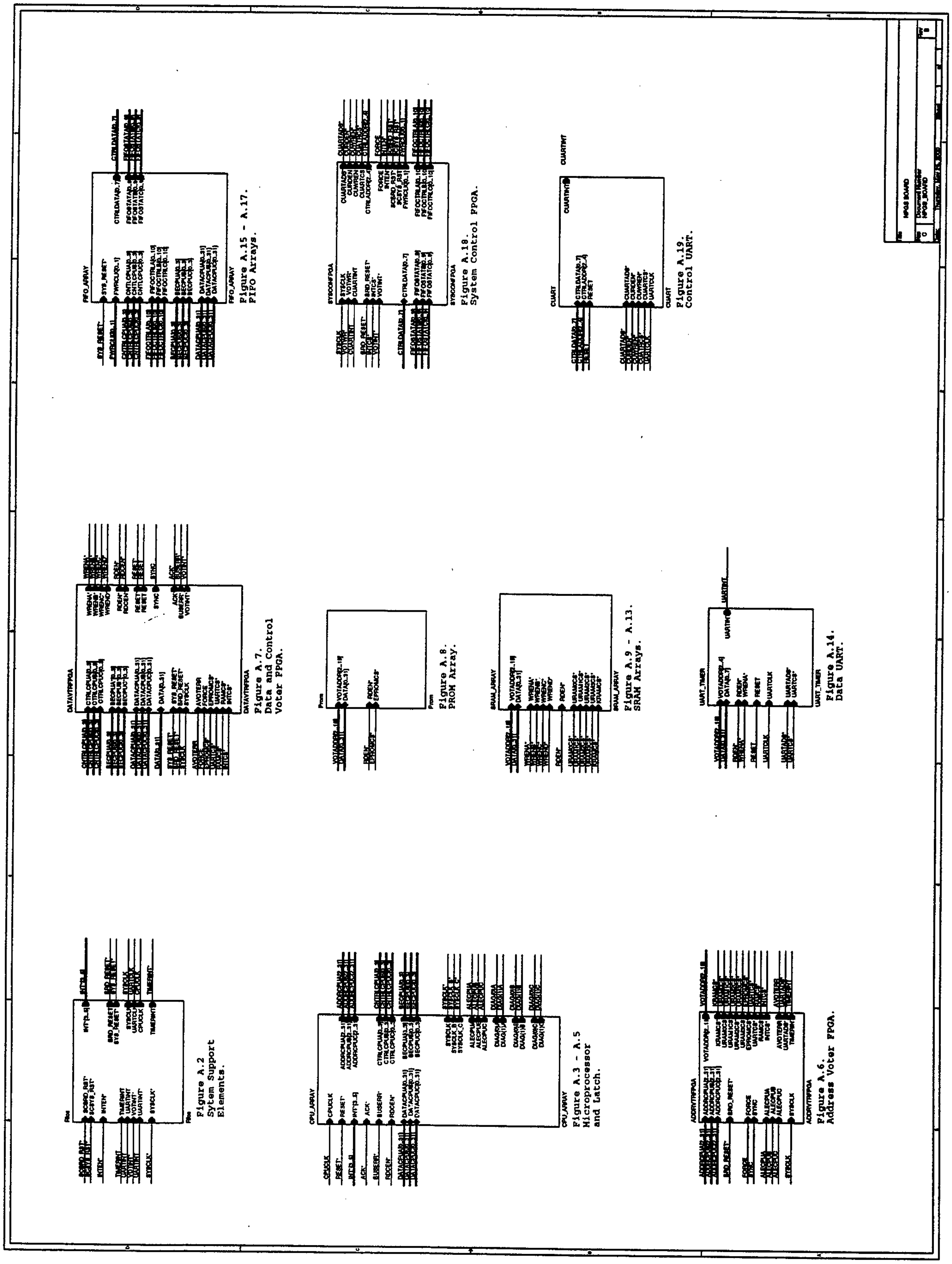
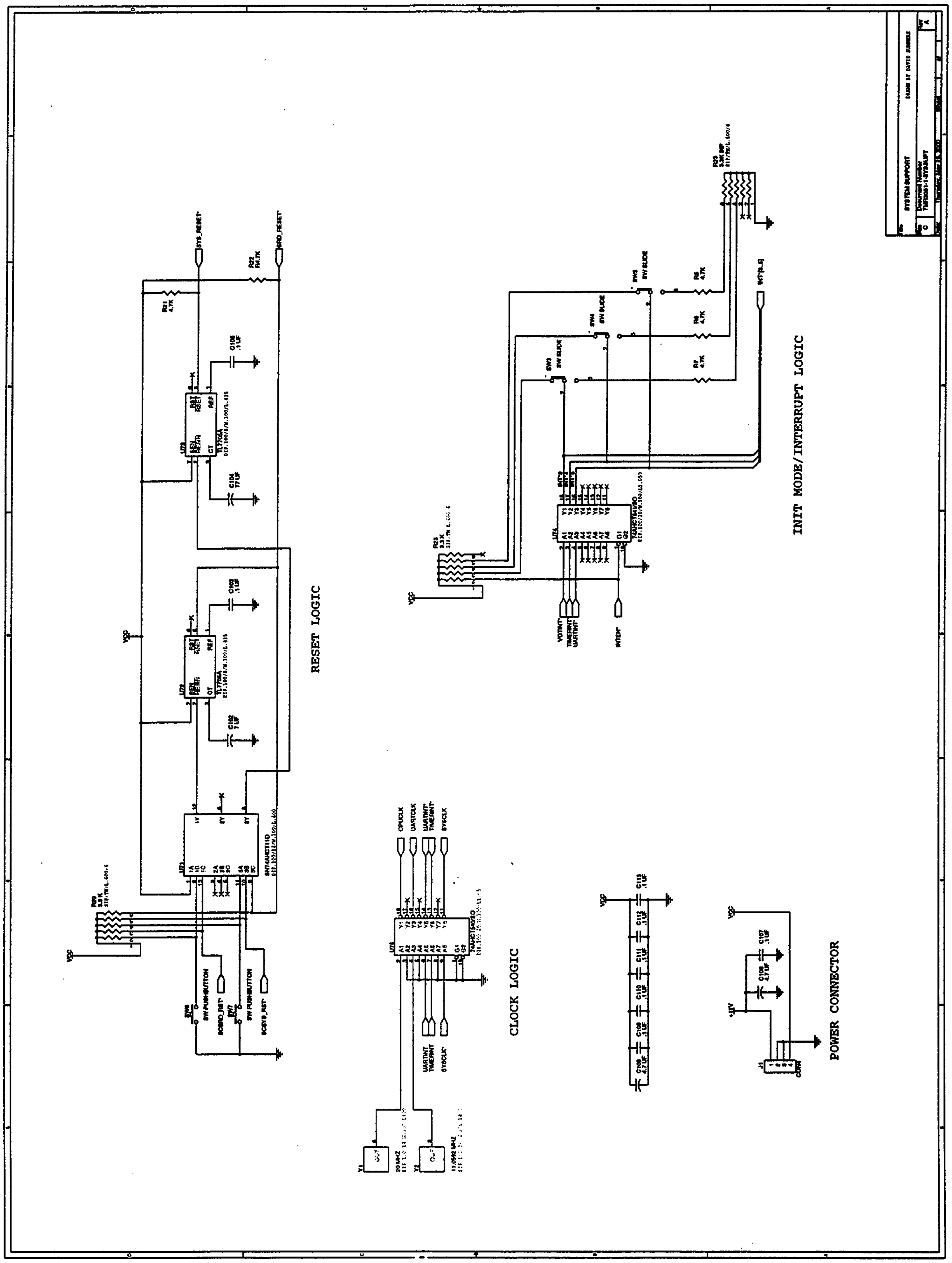
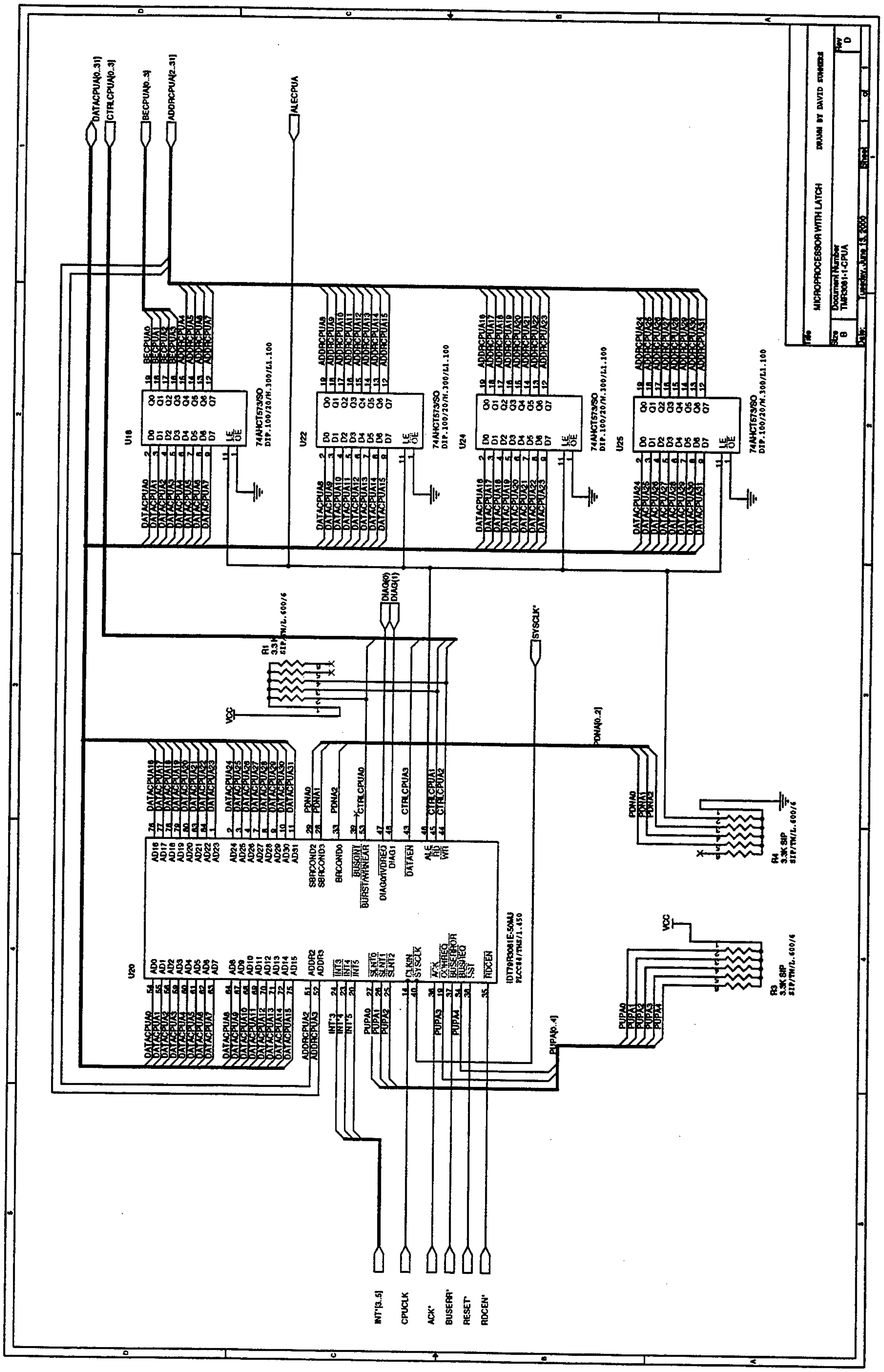
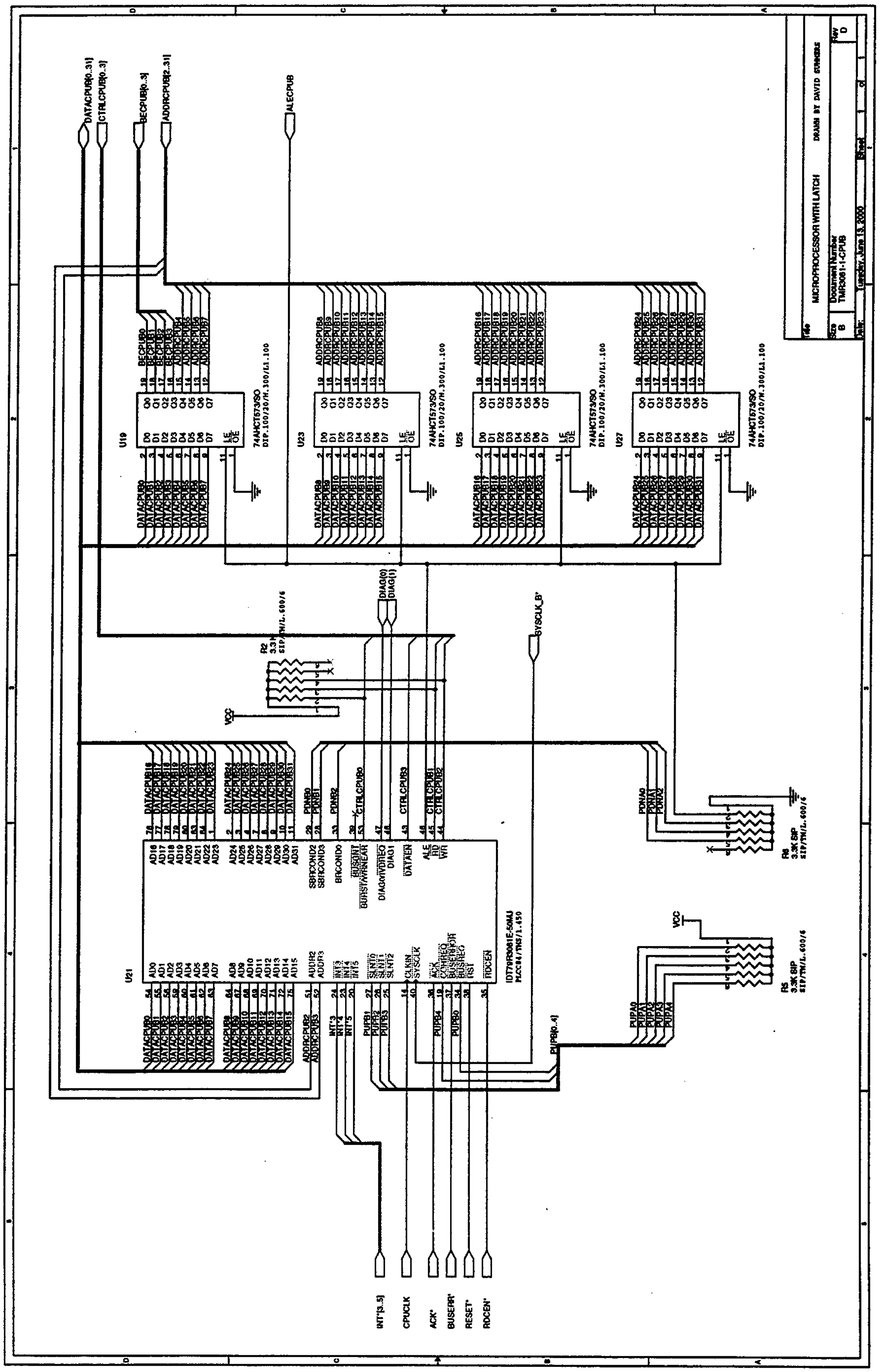


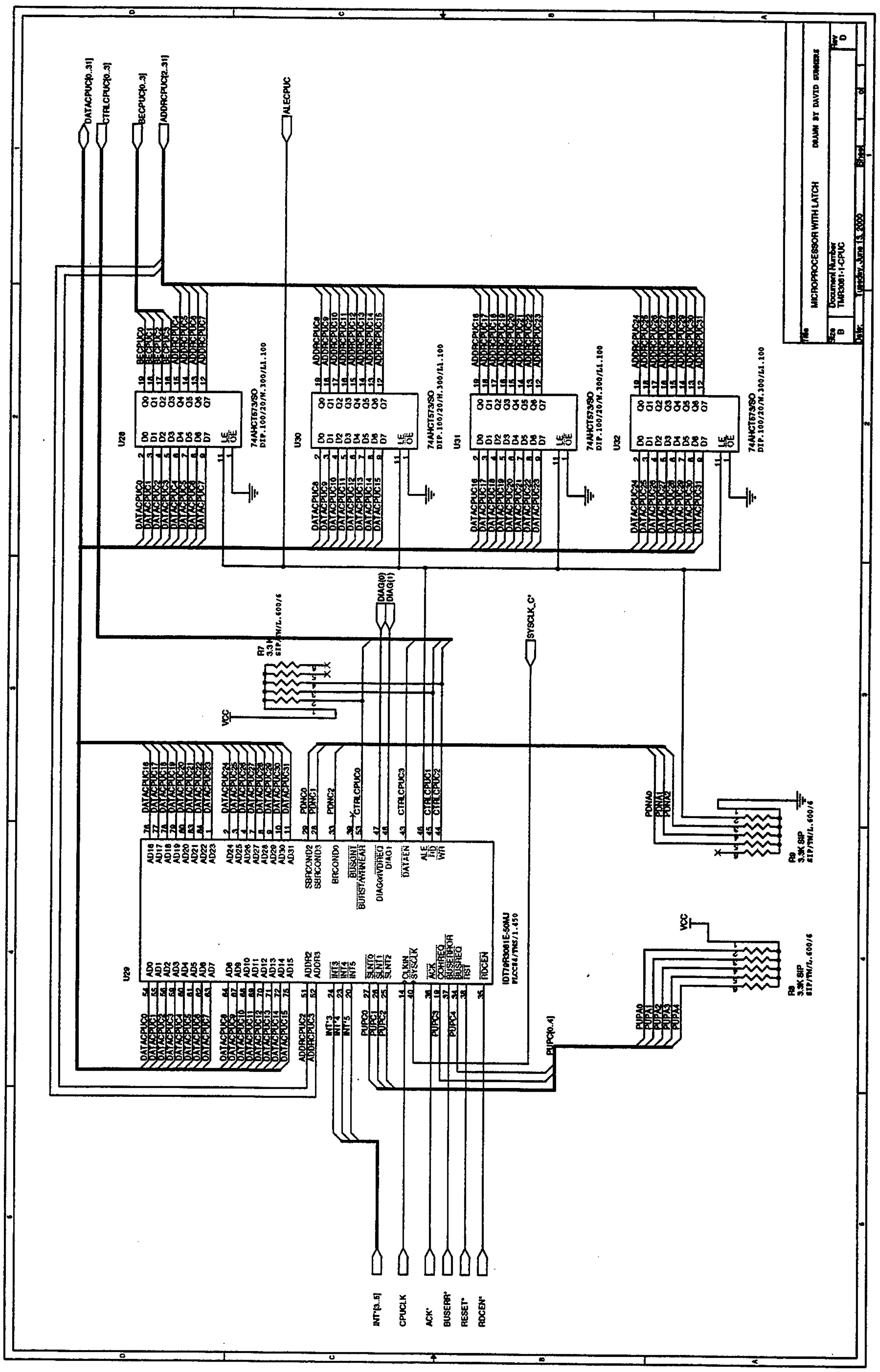
Figure A.2 Sytem Support Elements.
Figure A.3 - A.5 Microprocessor and Latch.
Figure A.6. Address Voter FPGA.
Figure A.7. Data and Control Voter FPGA.
Figure A.8. PROM Array.
Figure A.9 - A.13. SRAM Arrays.
Figure A.14. Data UART.
Figure A.15 - A.17. FIFO Arrays.
Figure A.18. System Control FPGA.
Figure A.19. Control UART.
NPGS BOARD


RESET LOGIC
INIT MODE/INTERRUPT LOGIC
CLOCK LOGIC
POWER CONNECTOR
SYS_RESET*
BRD_RESET*
VCC
SW PUSHBUTTON
SW SLIDE
CPUCLK
UARTCLK
UARTINT*
TIMERINT*
SYSCLK
SYSTEM SUPPORT

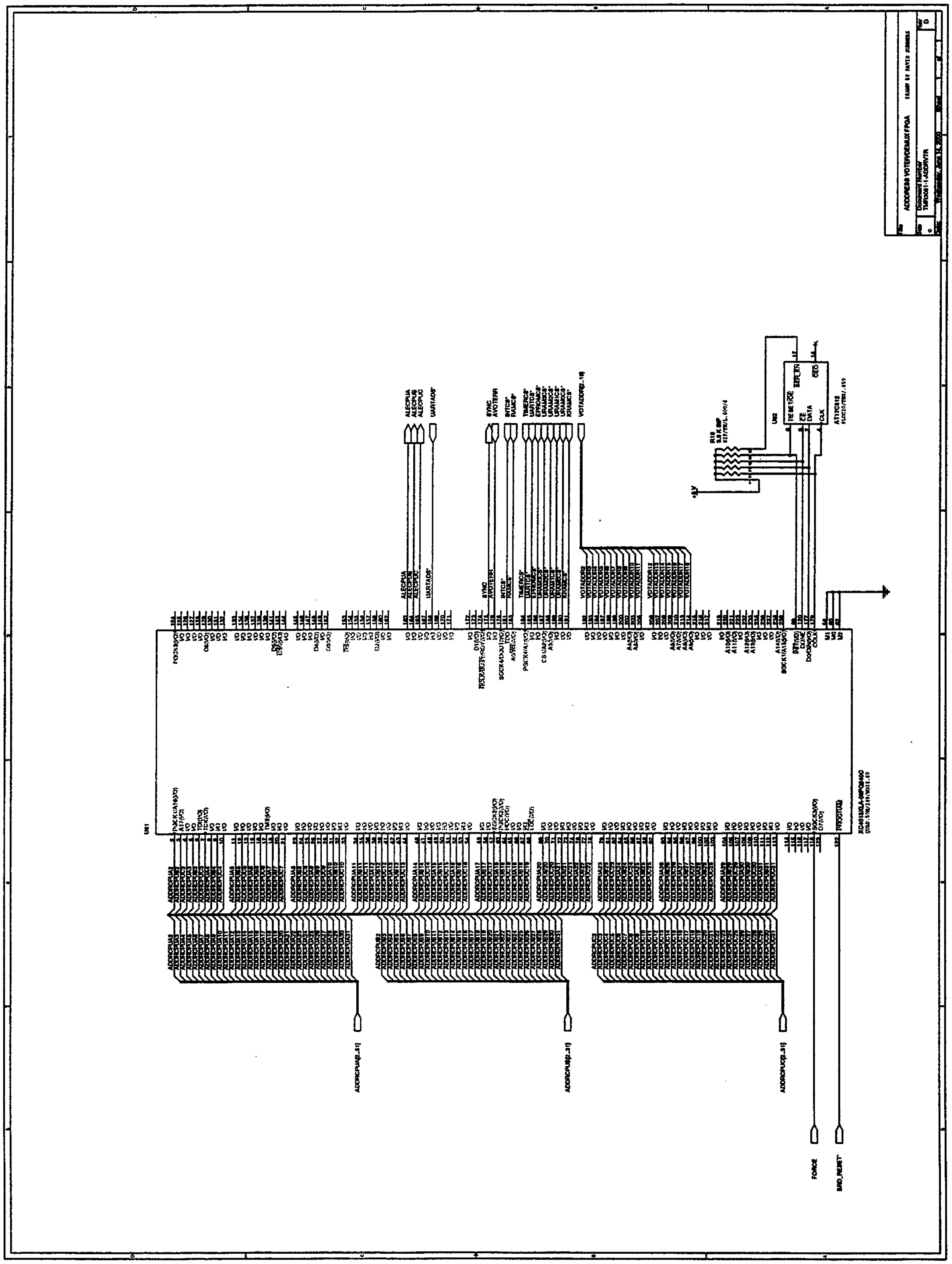


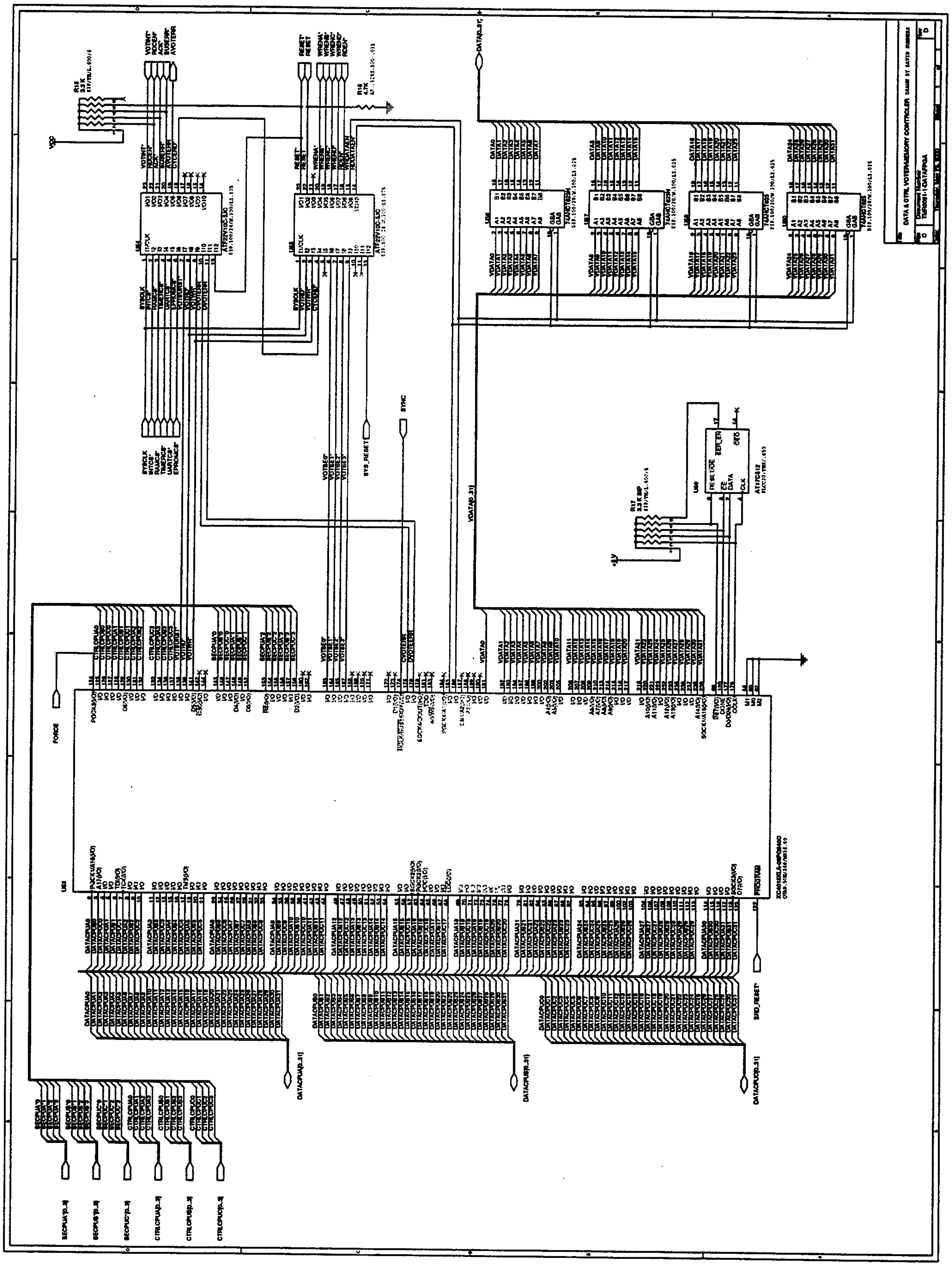
MICROPROCESSOR WITH LATCH
DRAWN BY DAVID SUMMERS
Document Number
TMR3081-1-CPUA
Size B
Rev D
Tuesday, June 13, 2000
Sheet 1 of 1
DATACPUA[0..31]
CTRLCPUA[0..3]
BECPUA[0..3]
ADDRCPUA[2..31]
ALECPUA
U18
U22
U24
U25
74AHCT573/SO
U20
IDT79R3081E-50MJ
DIAG(0)
DIAG(1)
SYSCLK*
R1 3.3K
R3 3.3K SIP
R4 3.3K SIP
PDNA[0..2]
PUPA[0..4]
VCC
INT*[3..5]
CPUCLK
ACK*
BUSERR*
RESET*
RDCEN*

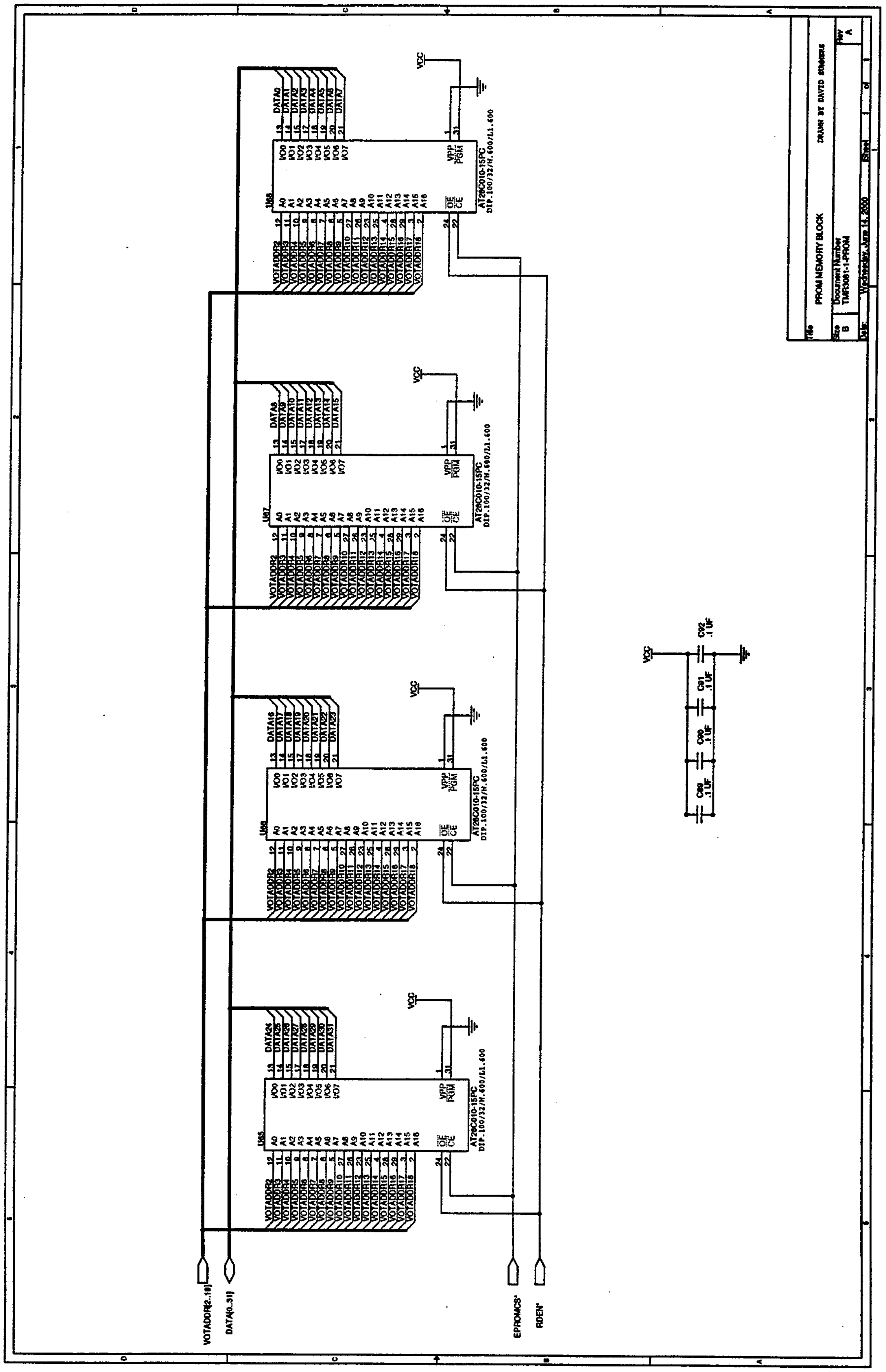

MICROPROCESSOR WITH LATCH
DRAWN BY DAVID SUNKEL
Document Number TMR3081-1-CPUB
Size B
Rev D
Tuesday, June 13, 2000
Sheet 1 of 1
DATACPUB[0..31]
CTRLCPUB[0..3]
BECPUB[0..3]
ADDRCPUB[2..31]
ALECPUB
DIAG(0)
DIAG(1)
SYSCLK_B*
U19
U23
U25
U27
74AHCT573/SO
U21
IDT79R3081E-50MJ
R2 3.3K
R5 3.3K SIP
R6 3.3K SIP
VCC
PUPB[0..4]
INT*[3..5]
CPUCLK
ACK*
BUSERR*
RESET*
RDCEN*

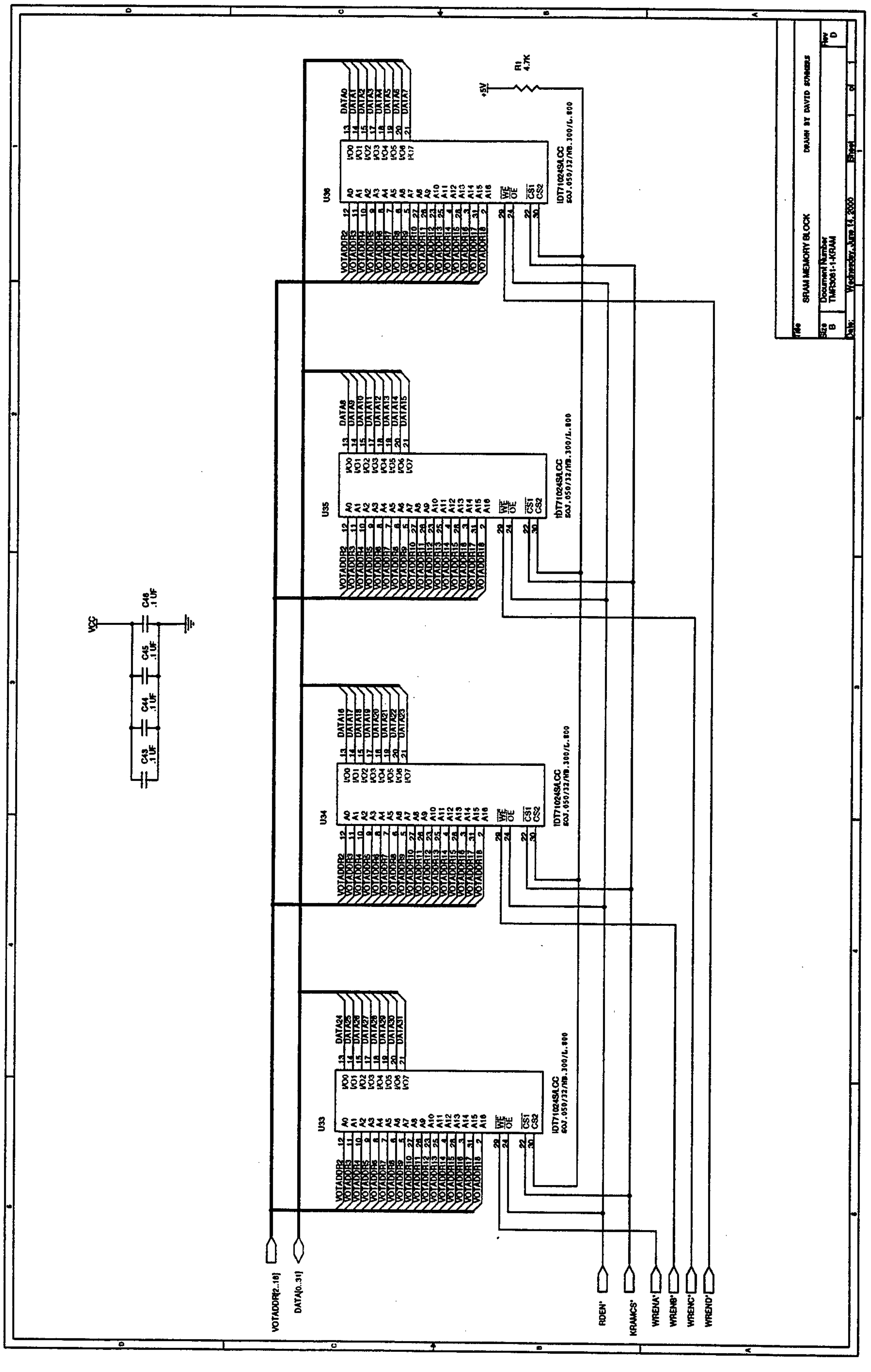


Title
MICROPROCESSOR WITH LATCH
DRAWN BY DAVID SUMMERS
Size B
Document Number TMR3081-1-CPUC
Rev D
Date: Tuesday, June 13, 2000
Sheet 1 of 1
DATACPUC[0..31]
CTRLCPUC[0..3]
BECPUC[0..3]
ADDRCPUC[2..31]
ALECPUC
U28
U30
U31
U32
74AHCT573/SO
DIP.100/20/W.300/L1.100
U29
IDT79R3081E-50MJ
PLCC84/TN5/1.450
R7
3.3K
R8
3.3K SIP
R9
3.3K SIP
VCC
DIAG(0)
DIAG(1)
SYSCLK_C*
PUPC[0..4]
INT*[3..5]
CPUCLK
ACK*
BUSERR*
RESET*
RDCEN*

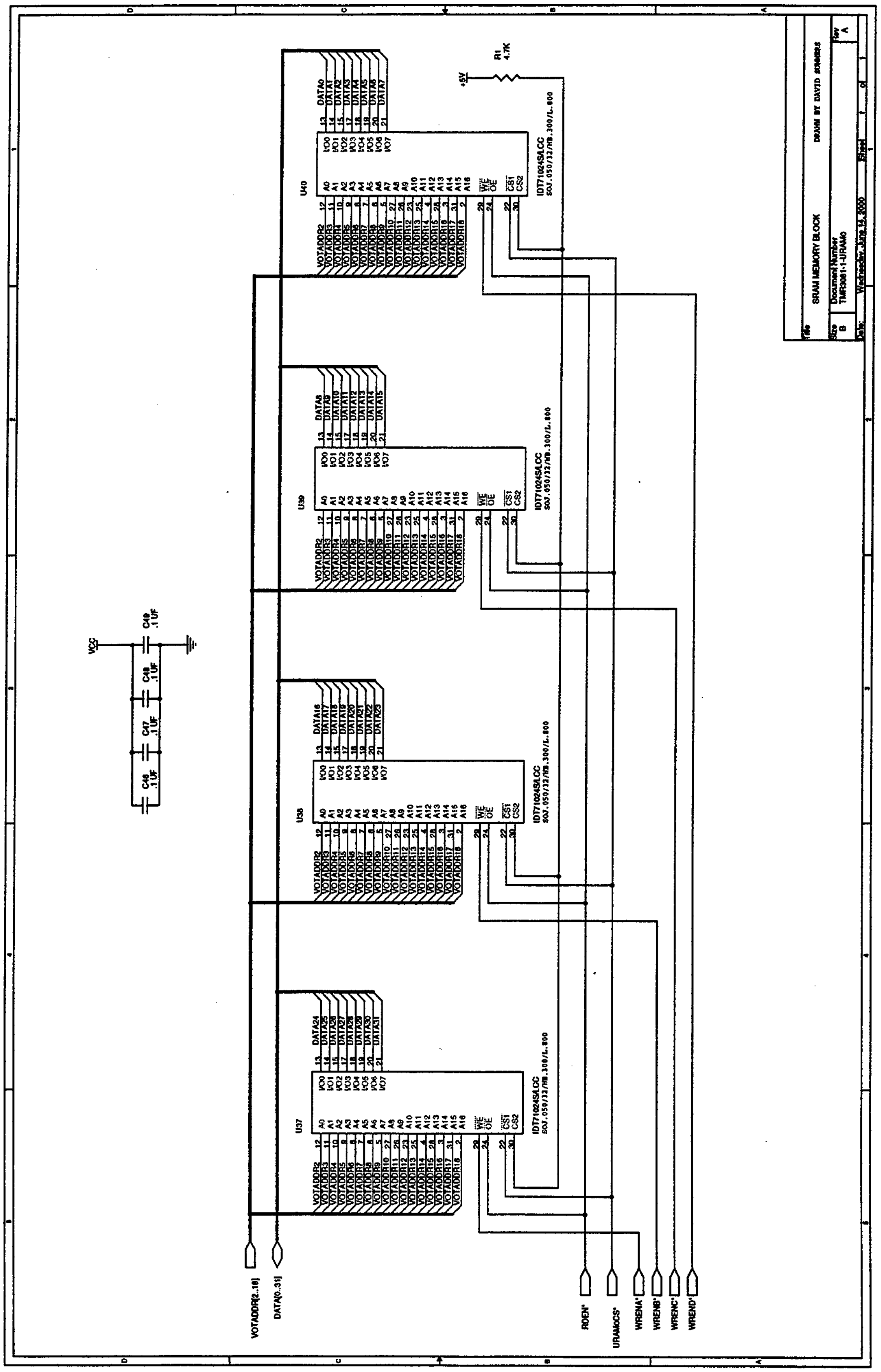

ADDRESS VOTER/DEMUX FPGA
DRAWN BY DAVID SUMMERS
Document Number
TMR0001-1-ADDRVTR
Rev
D
U61
U62
R18
ADDRCPUA[2..31]
ADDRCPUB[2..31]
ADDRCPUC[2..31]
ALECPUA
ALECPUB
ALECPUC
UARTADS*
SYNC
AVOTERR
INTCS*
RAMCS*
TIMERCS*
UARTCS*
EPROMCS*
VOTADDR[2..18]
FORCE
BRD_RESET*
+5V
AT17C512

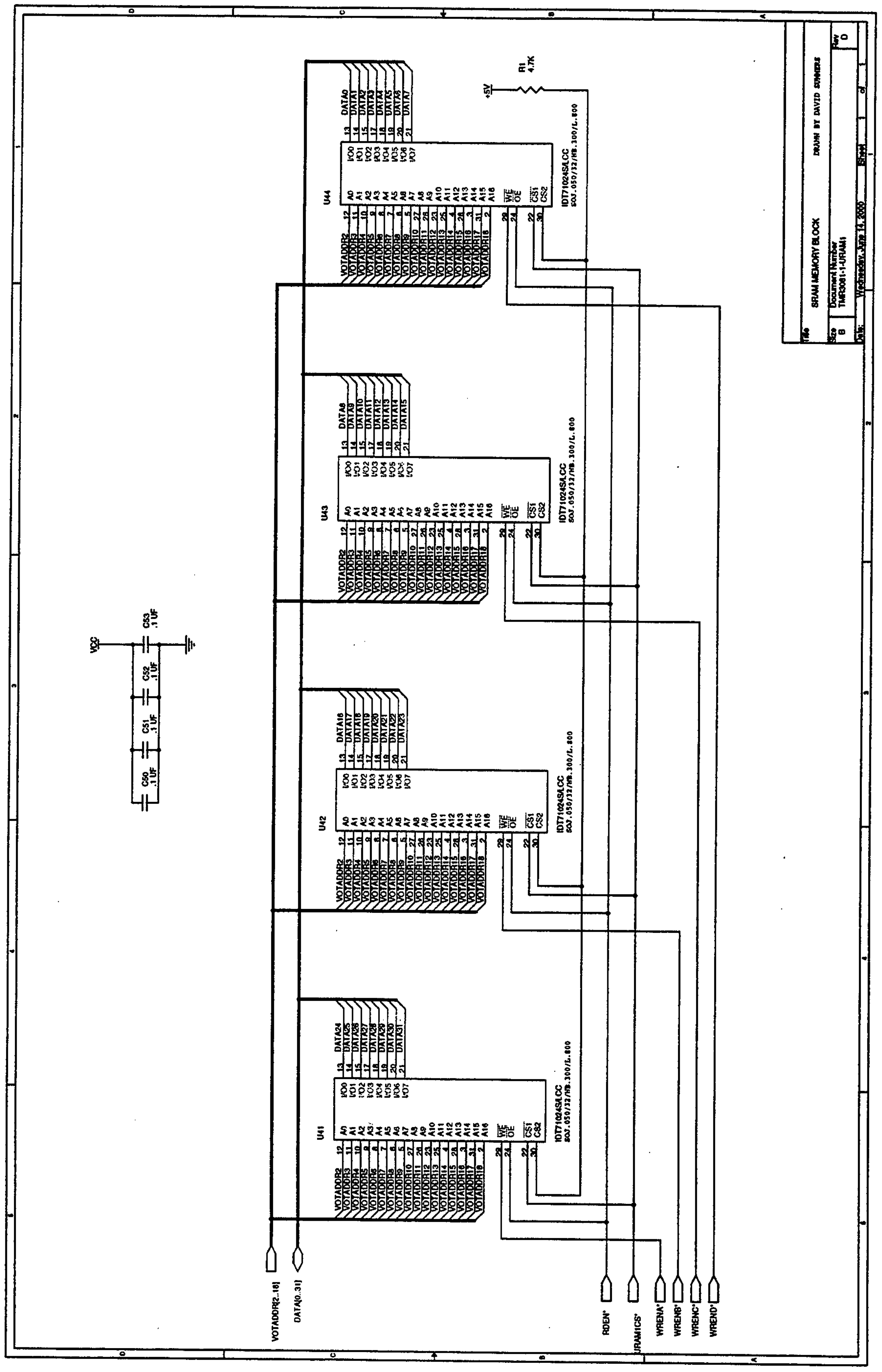

DATA & CTRL VOTER/MEMORY CONTROLLER
FORCE
SYNC
SYS_RESET*
BRD_RESET*
VDATA[0..31]
DATA[0..31]
DATACPUA[0..31]
DATACPUB[0..31]
DATACPUC[0..31]
SECPUA*[0..3]
SECPUB*[0..3]
SECPUC*[0..3]
CTRLCPUA[0..3]
CTRLCPUB[0..3]
CTRLCPUC[0..3]
VDD
+5V


PROM MEMORY BLOCK
DRAWN BY DAVID SUMMERS
Document Number
Wednesday, June 14, 2000
AT28C010-15PC
U65
U66
U67
U68
VCC
VOTADDR[2..18]
DATA[0..31]
EPROMCS*
RDEN*
.1 UF

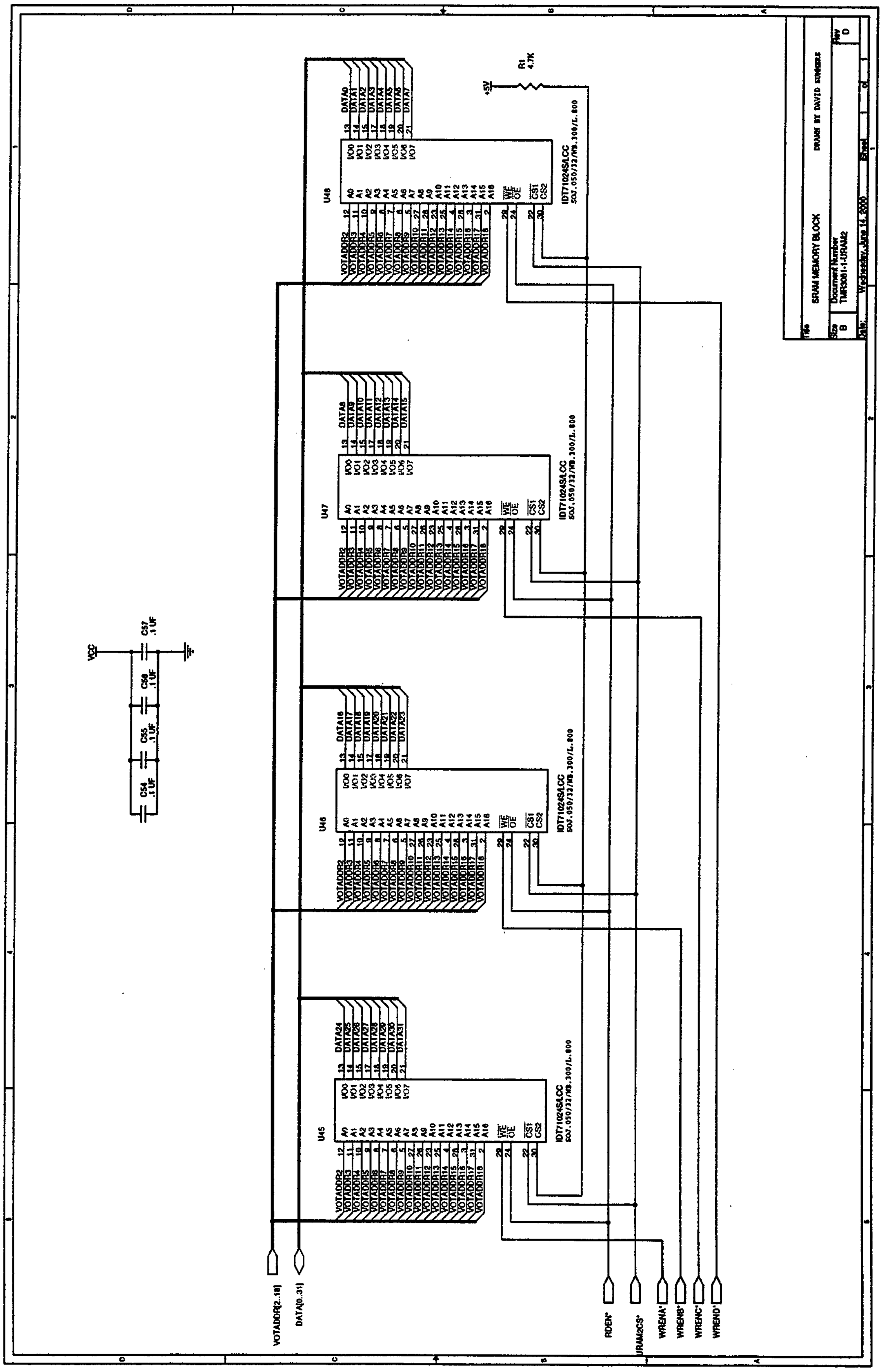


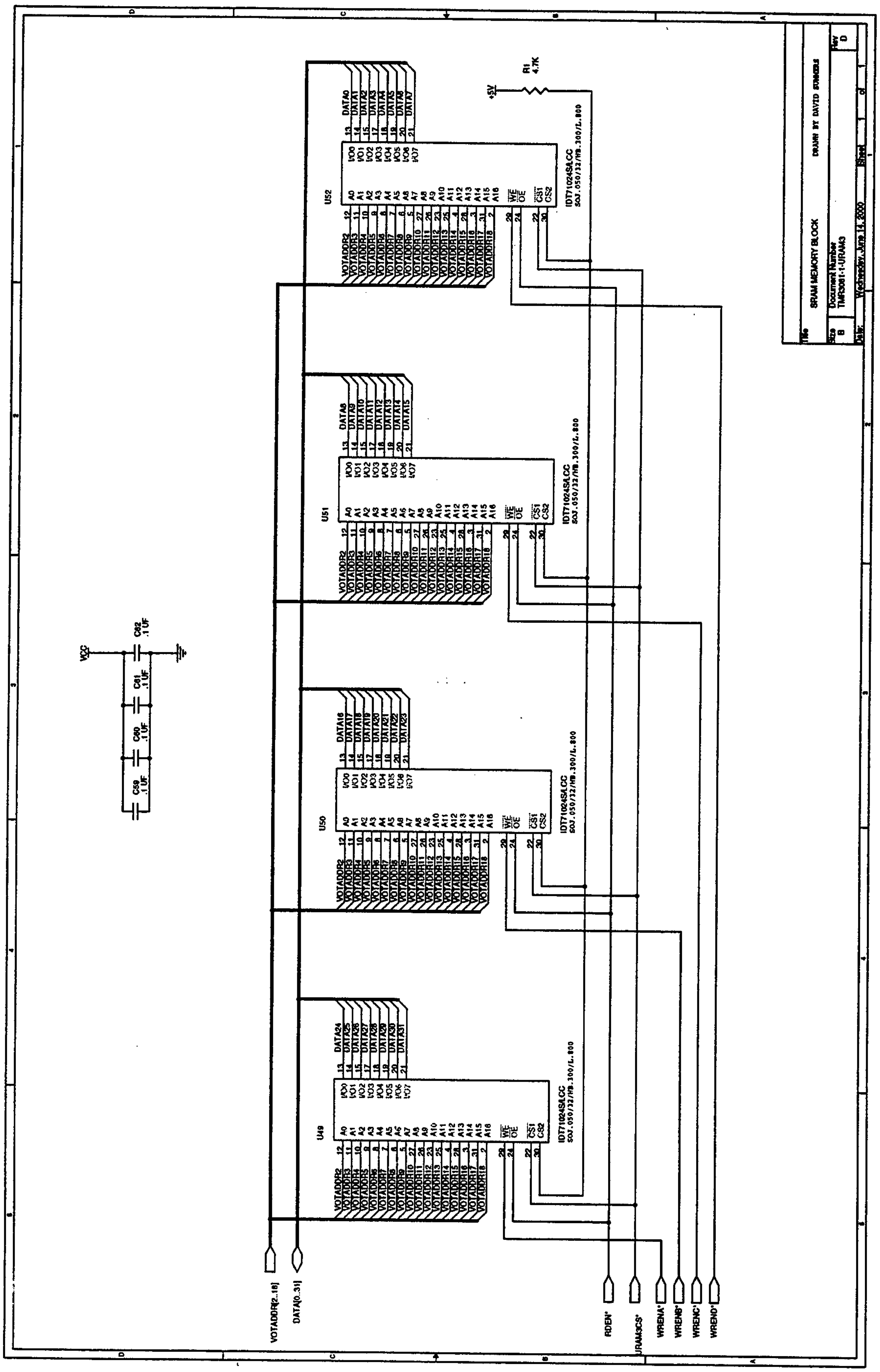
SRAM MEMORY BLOCK
DRAWN BY DAVID SOMMERS
Document Number TMR3061-1-KRAM
Size B
Rev D
Wednesday, June 14, 2000
Sheet 1 of 1
U33
U34
U35
U36
IDT71024S/LCC
SOJ.050/32/WB.300/L.800
R1 4.7K
+5V
VCC
C43 .1UF
C44 .1UF
C45 .1UF
C46 .1UF
VOTADDR[2..18]
DATA[0..31]
RDEN*
KRAMCS*
WRENA*
WRENB*
WRENC*
WREND*


SRAM MEMORY BLOCK
DRAWN BY DAVID SUMMERS
Document Number TMR3061-1-URAM0
Size B
Rev A
Date: Wednesday, June 14, 2000
Sheet 1 of 1
U37
U38
U39
U40
IDT71024S/LCC
R1 4.7K
+5V
VCC
C46 .1UF
C47 .1UF
C48 .1UF
C49 .1UF
VOTADDR[2..18]
DATA[0..31]
RDEN*
URAMOCS*
WRENA*
WRENB*
WRENC*
WREND*

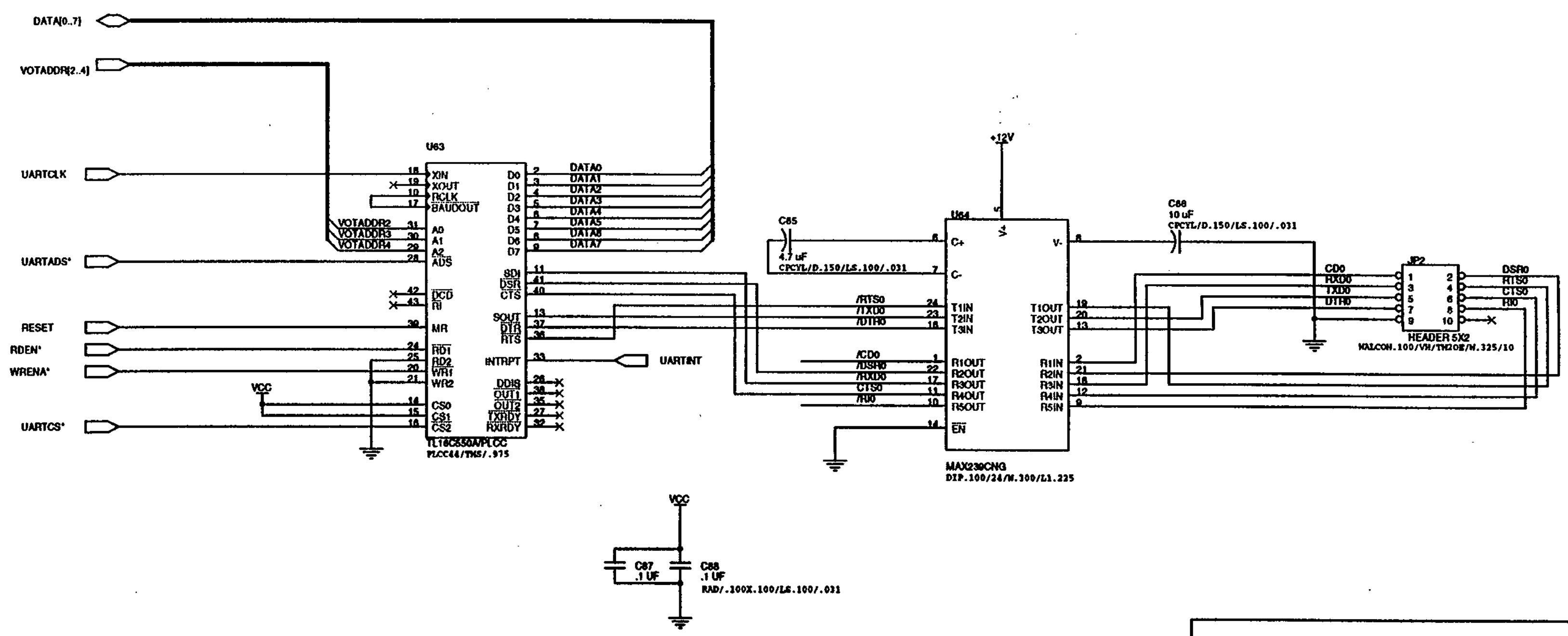


SRAM MEMORY BLOCK
DRAWN BY DAVID SUMMERS
Document Number
TMR3081-1-URAM1
Wednesday, June 14, 2000
U41
U42
U43
U44
IDT71024S/LCC
R1
4.7K
+5V
VCC
C50
C51
C52
C53
.1 UF
VOTADDR[2..18]
DATA[0..31]
RDEN*
URAM1CS*
WRENA*
WRENB*
WRENC*
WREND*


SRAM MEMORY BLOCK
DRAWN BY DAVID SUNDERS
Document Number
TMR3061-URAM2
Size B
Rev D
Date: Wednesday, June 14, 2000
U45
U46
U47
U48
IDT71024S/LCC
SOJ.050/32/WB.300/L.800
VOTADDR[2..18]
DATA[0..31]
RDEN*
URAM2CS*
WRENA*
WRENB*
WRENC*
WREND*
VCC
C54 .1 UF
C55 .1 UF
C56 .1 UF
C57 .1 UF
+5V
R1 4.7K



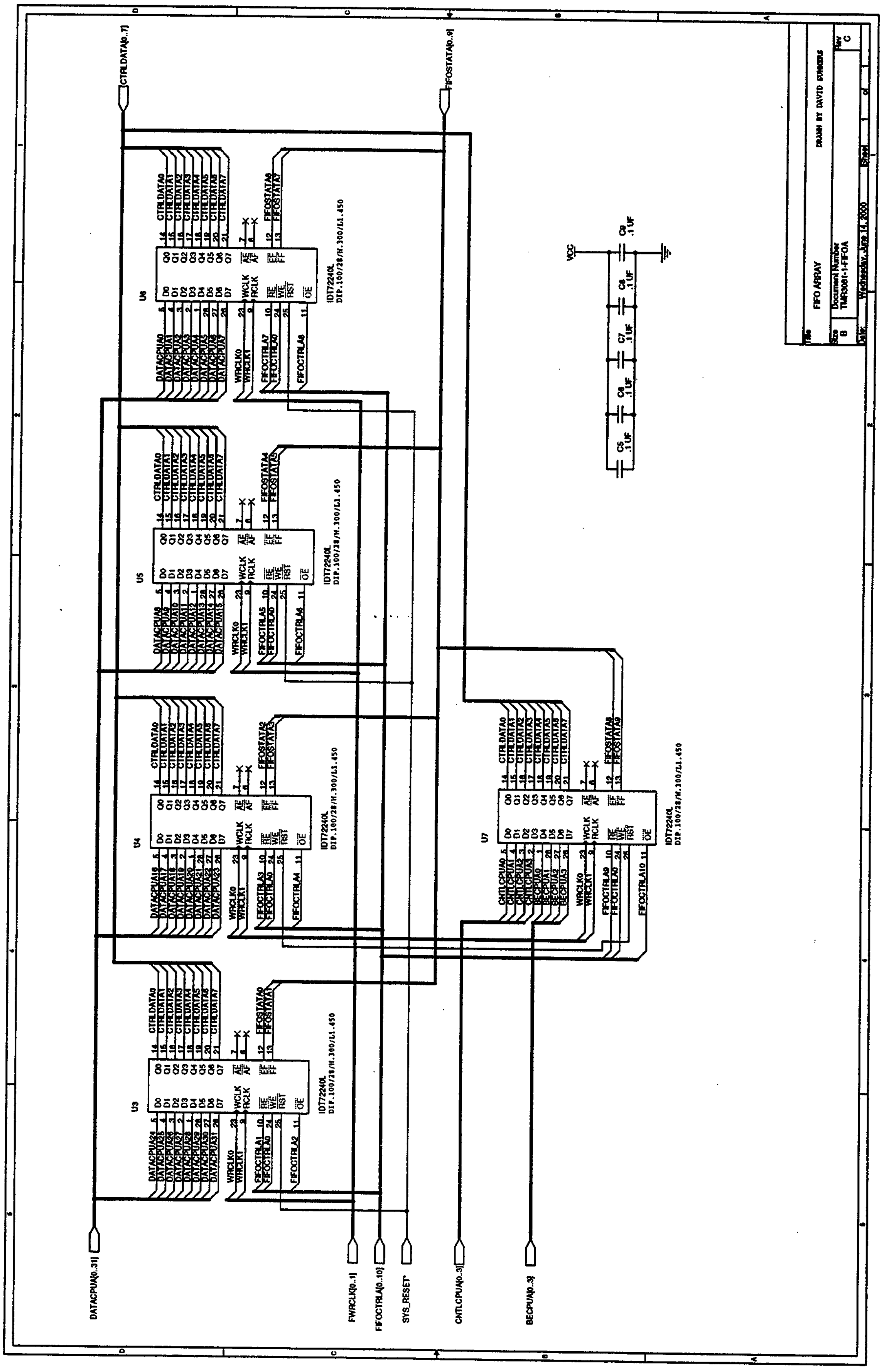
SRAM MEMORY BLOCK
DRAWN BY DAVID SUNKERS
Document Number
TMR3081-1-URAM3
Size B
Rev D
Date: Wednesday, June 14, 2000
Sheet 1 of 1
U49
U50
U51
U52
IDT71024S/LCC
SOJ.050/32/WB.300/L.800
R1
4.7K
+5V
VCC
C59 .1UF
C60 .1UF
C61 .1UF
C62 .1UF
VOTADDR[2..18]
DATA[0..31]
RDEN*
URAM3CS*
WRENA*
WRENB*
WRENC*
WREND*

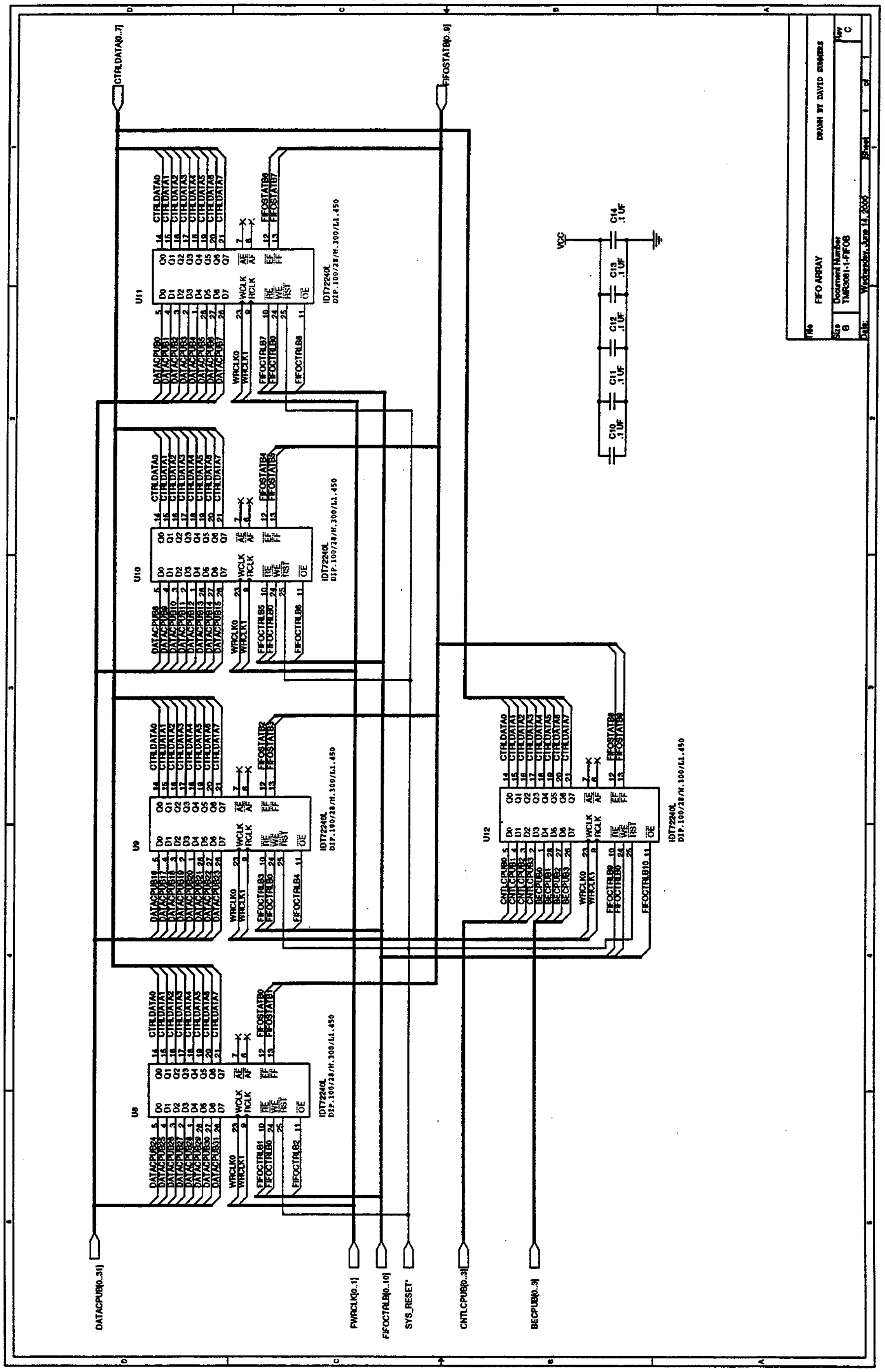


Title
UART - TIMER (I/O BLOCK)
DRAWN BY DAVID SUMMERS

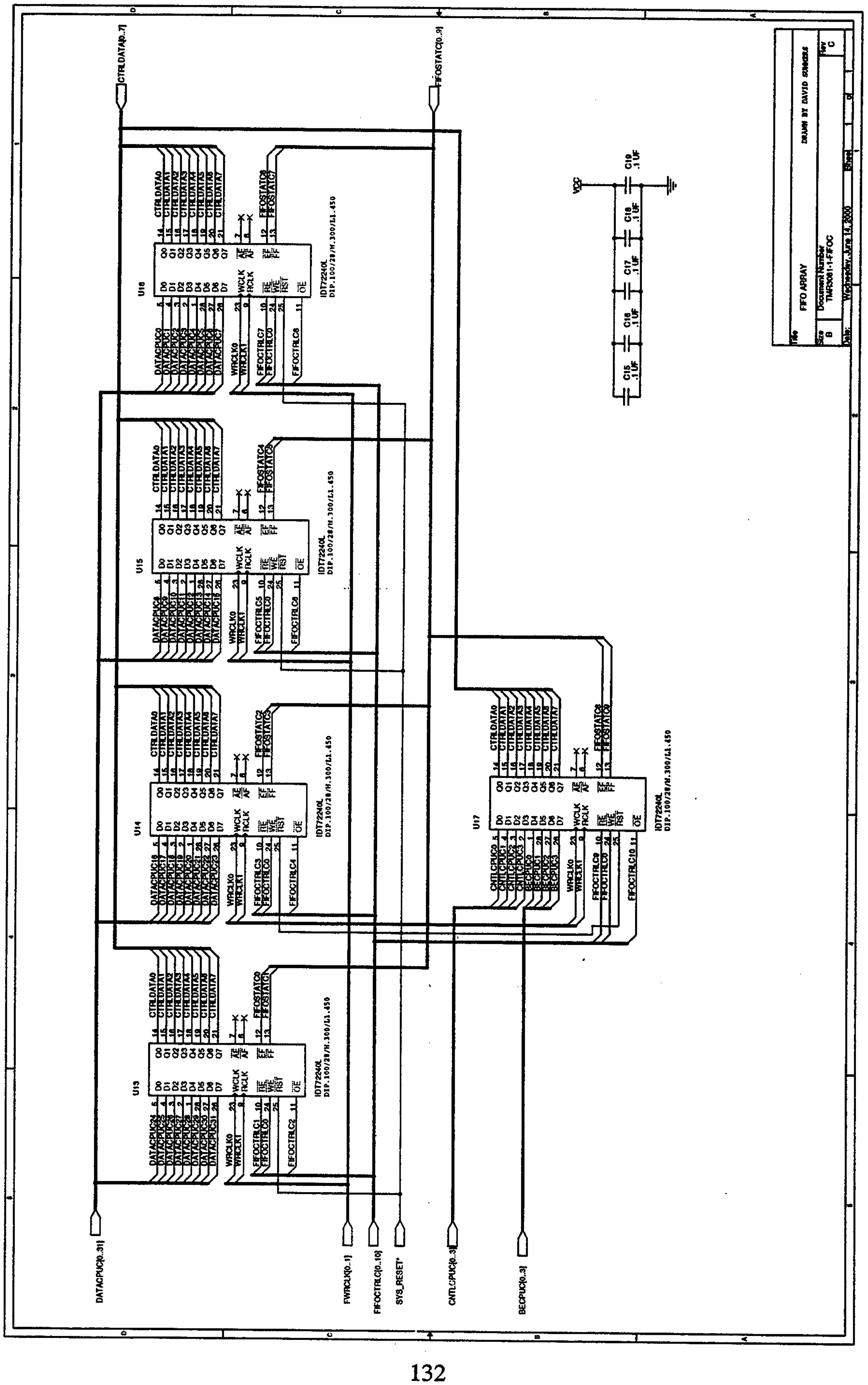
Size	Document Number	Rev
B	TRM3081-1-UART/TIMER	C

Date: Wednesday, June 14, 2000 Sheet 1 of 1

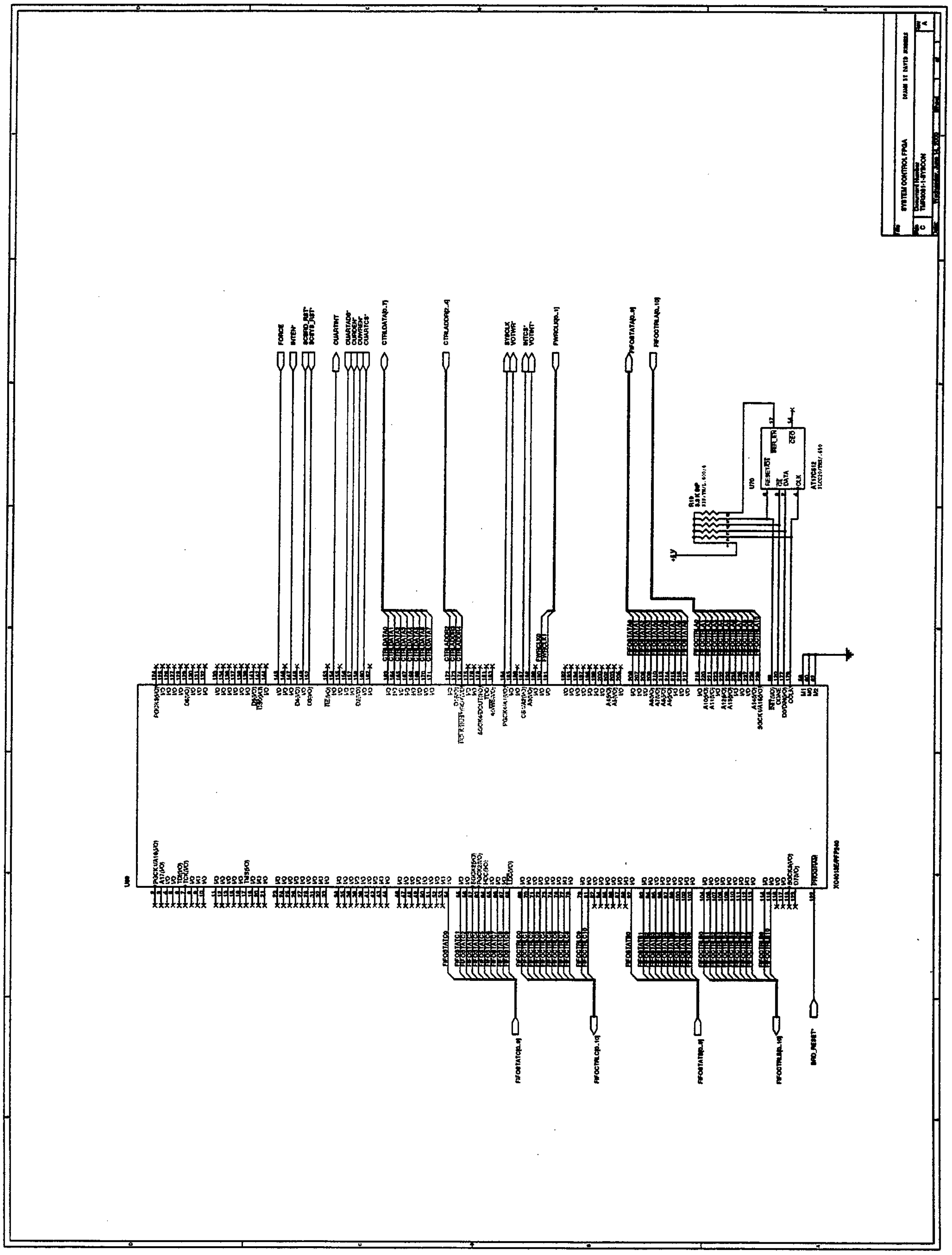

FIFO ARRAY
Document Number TMR3081-1-FIFOA
Size B
Rev C
DRAWN BY DAVID SUMMERS
Date: Wednesday, June 14, 2000
Sheet 1 of 1
U3
U4
U5
U6
U7
IDT72240L
DIP.100/28/W.300/L1.450
CTRLDATA[0..7]
FIFOSTATA[0..9]
DATACPUA[0..31]
FWRCLK[0..1]
FIFOCTRLA[0..10]
SYS_RESET
CNTLCPUA[0..3]
BECPUA[0..3]
VCC
C5 .1 UF
C6 .1 UF
C7 .1 UF
C8 .1 UF
C9 .1 UF

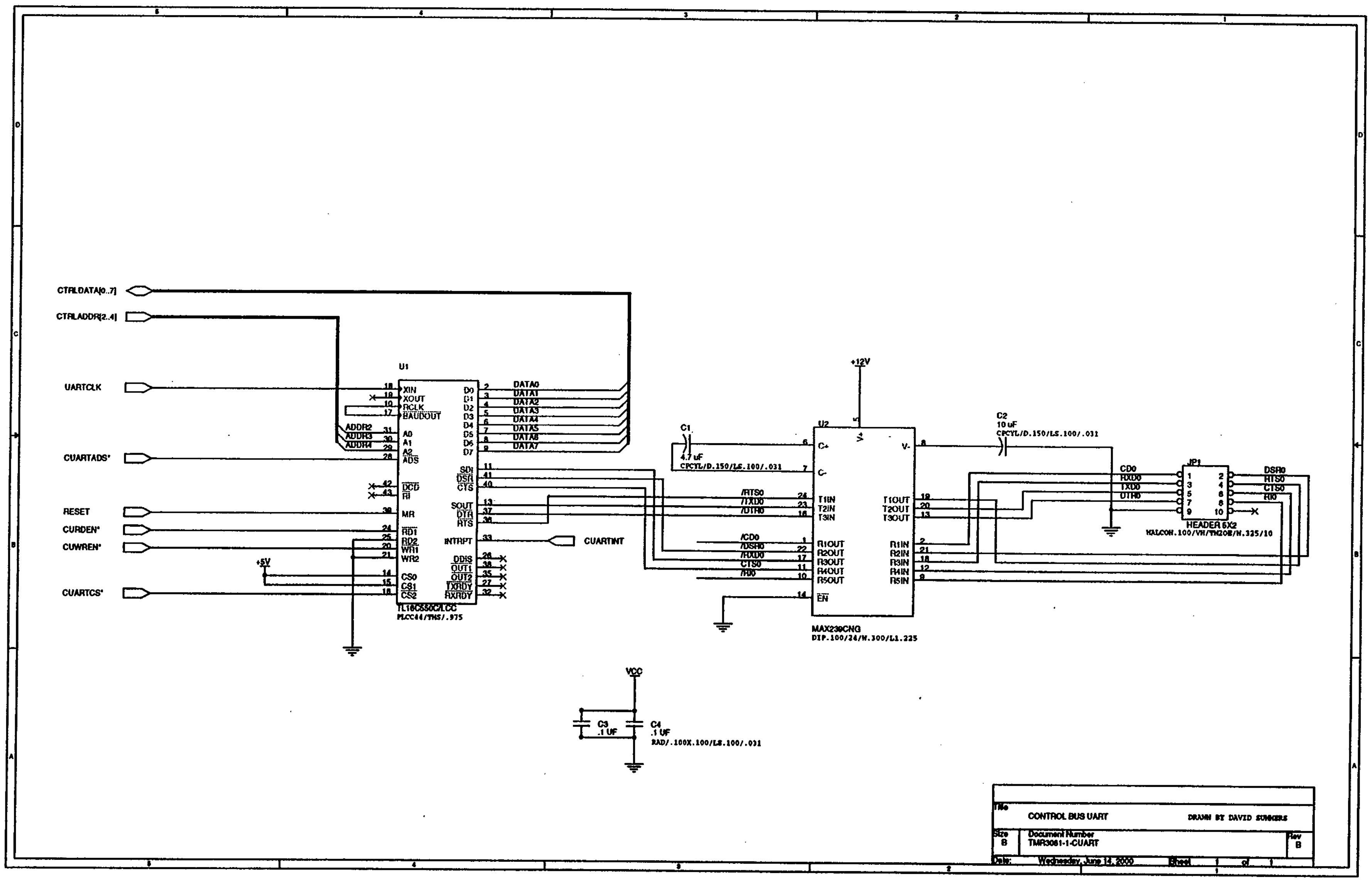


CTRLDATA[0..7]
FIFOSTATB[0..9]
DATACPUB[0..31]
FWRCLK[0..1]
FIFOCTRLB[0..10]
SYS_RESET*
CNTLCPUB[0..3]
BECPUB[0..3]
U8
U9
U10
U11
U12
IDT72240L
DIP.100/28/H.300/L1.450
VCC
C10 .1 UF
C11 .1 UF
C12 .1 UF
C13 .1 UF
C14 .1 UF
DRAWN BY DAVID SUMMERS
Title FIFO ARRAY
Size B
Document Number TMR3081-1-FIFO8
Rev C
Date: Wednesday, June 14, 2000
Sheet 1 of 1



FIFO ARRAY
DRAWN BY DAVID SIMMONS
Document Number
TMR3081-1-FIFOC
Size B
Rev C
Date: Wednesday, June 14, 2000
Sheet 1 of 1
U13
U14
U15
U16
U17
IDT72240L
DIP.100/28/W.300/L1.450
C15 .1 UF
C16 .1 UF
C17 .1 UF
C18 .1 UF
C19 .1 UF
VCC
CTRLDATA[0..7]
FIFOSTATC[0..9]
DATACPUC[0..31]
FWRCLK[0..1]
FIFOCTRLC[0..10]
SYS_RESET*
CNTLCPUC[0..3]
BECPUC[0..3]
WRCLK0
WRCLK1
WCLK
RCLK
AE
AF
EF
FF
RE
WE
RST
OE


SYSTEM CONTROL FPGA
TLRC011-1-SYSCON
U70
AT17C512
R19
+5V
FORCE
INTEN*
CTRLDATA[0..7]
CTRLADDR[0..4]
SYSCLK
FIFODATA[0..9]
FIFOCTRLA[0..10]
FIFOSTATC[0..9]
FIFOCTRLC[0..10]
FIFOSTATB[0..9]
FIFOCTRLB[0..10]
BRD_RESET*
XC4010E/PF240


CTRLDATA[0..7]
CTRLADDR[2..4]
UARTCLK
CUARTADS*
RESET
CURDEN*
CUWREN*
CUARTCS*
+5V
U1
XIN
XOUT
RCLK
BAUDOUT
A0
A1
A2
ADS
DCD
RI
MR
RD1
RD2
WR1
WR2
CS0
CS1
CS2
D0
D1
D2
D3
D4
D5
D6
D7
SDI
DSR
CTS
SOUT
DTR
RTS
INTRPT
DDIS
OUT1
OUT2
TXRDY
RXRDY
TL16C550C/LCC
PLCC44/TMS/.975
ADDR2
ADDR3
ADDR4
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
CUARTINT
C1
4.7 uF
CPCYL/D.150/LS.100/.031
/RTS0
/TXD0
/DTR0
/CD0
/DSR0
/RXD0
CTS0
/RI0
+12V
U2
C+
C-
V+
V-
T1IN
T2IN
T3IN
R1OUT
R2OUT
R3OUT
R4OUT
R5OUT
EN
T1OUT
T2OUT
T3OUT
R1IN
R2IN
R3IN
R4IN
R5IN
MAX239CNG
DIP.100/24/W.300/L1.225
C2
10 uF
CPCYL/D.150/LS.100/.031
CD0
RXD0
TXD0
DTR0
JP1
DSR0
RTS0
CTS0
RI0
HEADER 6X2
MALCON.100/VH/TM20B/W.325/10
VCC
C3
.1 UF
C4
.1 UF
RAD/.100X.100/LS.100/.031
Title
CONTROL BUS UART
DRAWN BY DAVID SUMMERS
Size
B
Document Number
TMR3081-1-CUART
Rev
B
Date:
Wednesday, June 14, 2000
Sheet
1
of
1

APPENDIX B. WINCUPL FILES

When using programmable devices, such as Programmable Logic Devices (PLDs) and Field Programmable Gate Arrays (FPGAs), files must be written that instruct the device how to function when signals are applied to it. For the TMR R3081 design, a software program known as WinCUPL was used to write and compile the design files for the PLDs. WinCUPL also offers a simulation tool that allows the designer to test the programming files before the device is actually programmed in order to determine a specified set of test vectors produce the expected outputs. Sections A and B of this appendix contain the program file and the simulation file, respectively, for the Memory Control PLD that was presented in Chapter IV. Sections C and D contain the program and simulation file for the Memory Enable PLD that was also presented in Chapter IV.

1. MEMORY CONTROL PLD PROGRAM FILE

```
Name     MemCont ;
PartNo   ATF22V10C-7PC ;
Date     4/30/00 ;
Revision 01 ;
Designer David Summers ;
Company  NPS ;
Assembly TMR R3081 ;
Location U54 ;
Device   p22v10 ;

/* ******************    CONSTANT DEFINITIONS     ******************* */
$DEFINE LOW  'B'0
$DEFINE HIGH 'B'1

/* ***********************      INPUT PINS     *********************** */
PIN   1 = SYSCLK;      /*  SYSCLK FR CPUA                    */
PIN   2 = INTCSN;      /*  INTERRUPT CHIP SELECT             */
PIN   3 = RAMCSN;      /*  SRAM CHIP SELECT                  */
PIN   4 = TIMERCSN;    /*  TIMER CHIP SELECT                 */
PIN   5 = UARTCSN;     /*  UART CHIP SELECT                  */
PIN   6 = EPROMCSN;    /*  EPROM CHIP SELECT                 */
PIN   7 = VOTBURSTN;   /*  VOTED BURST READ|WRITE NEAR       */
PIN   8 = VOTRDN;      /*  VOTED READ                        */
PIN   9 = VOTWRN;      /*  VOTED WRITE                       */
PIN  10 = CVOTERR;     /*  CONTROL VOTER ERROR               */
PIN  11 = DVOTERR;     /*  DATA VOTER ERROR                  */
PIN  13 = RESETN;      /*  SYNCHRONOUS SYSTEM RESET          */
PIN  19 = AVOTERR;     /*  ADDRESS VOTER ERROR               */
```

```
/* ***********************    OUTPUT PINS     *********************** */
PIN  [14,15,16,17] = [COUNT3..0];    /*  WAIT STATE GENERATOR        */
PIN  18 = CYCENDN;                   /*  CYCLE END SIGNAL            */
PIN  20 = BUSERRN;                   /*  BUS ERROR SIGNAL TO CPU     */
PIN  21 = ACKN;                      /*  ACK SIGNAL TO CPU           */
PIN  22 = RDCENN;                    /*  READ CLOCK ENABLE TO CPU    */
PIN  23 = VOTINTN;                   /*  VOTER INTERRUPT TO CPU      */

/* ***********************  LOGIC EQUATIONS   *********************** */

/* ****************************************************************** */
/*                                                                    */
/*                     WAIT STATE COUNTER COUNT[3..0]                 */
/*                                                                    */
/* THE PURPOSE OF THE WAIT STATE COUNTER IS TO PROVIDE TIMING         */
/* REFERENCE FOR THE MEMORY CONTROLLER SIGNALS.  IT STARTS COUNTING   */
/* WHEN A READ OR WRITE CYCLE IS INITIATED AND RESETS WHEN THERE IS A */
/* RESET OR CYCLE ENDSIGNAL.  THE COUNTER USES THE SYSCLK AS ITS      */
/* REFERENCE CLOCK.                                                   */
/* ****************************************************************** */

COUNT0.D = RESETN & CYCENDN & (!VOTRDN # !VOTWRN) &
             (COUNT0 $ HIGH);
COUNT1.D = RESETN & CYCENDN & (!VOTRDN # !VOTWRN) &
             (COUNT1 $ COUNT0);
COUNT2.D = RESETN & CYCENDN & (!VOTRDN # !VOTWRN) &
             (COUNT2 $ (COUNT1 & COUNT0));
COUNT3.D = RESETN & CYCENDN & (!VOTRDN # !VOTWRN) &
             (COUNT3 $ (COUNT2 & COUNT1 & COUNT0));

COUNT0.OE=HIGH;
COUNT1.OE = HIGH;
COUNT2.OE = HIGH;
COUNT3.OE = HIGH;

COUNT0.AR = LOW;
COUNT1.AR = LOW;
COUNT2.AR = LOW;
COUNT3.AR = LOW;

COUNT0.SP = LOW;
COUNT1.SP = LOW;
COUNT2.SP = LOW;
COUNT3.SP = LOW;

FIELD CNTR = [COUNT3, COUNT2, COUNT1, COUNT0];

/* ****************************************************************** */
/*                                                                    */
/*                        VOTER INTERRUPT SIGNAL                      */
/*                                                                    */
/* THE VOTER INTERRUPT SIGNAL INFORMS THE CPU WHEN A MISCOMPARE       */
/* HAPPENS IN ONE OF THE FPGAS.  THE SIGNAL IS HELD UNTIL A INTCSN    */
/* IS GENERATED BY THE ADDRESS DECODER.                               */
/* ****************************************************************** */

VOTINTN.D = !((AVOTERR # CVOTERR # DVOTERR # !VOTINTN) & INTCSN);
VOTINTN.AR = LOW;
VOTINTN.SP = LOW;
```

```
/* ****************************************************************** */
/*                                                                    */
/*                         CYCLE END SIGNAL                           */
/*                                                                    */
/* THE PURPOSE OF THE CYCLE END SIGNAL IS TO SIGNAL THE END OF THE    */
/* OF THE CURRENT BUS CYCLE IN ORDER TO RESET THE COUNTER.  THIS      */
/* RESETS ITSELF BY INCLUDING A REFERENCE TO ITSELF IN THE EQUATIONS. */
/* ****************************************************************** */

CYCENDN.D = !(RESETN & CYCENDN & (
                (!RAMCSN & (CNTR : 'H'0) & !VOTRDN & VOTBURSTN)
              # (!RAMCSN & (CNTR : 'H'6) & !VOTRDN & !VOTBURSTN)
              # (!RAMCSN & (CNTR : 'H'0) & !VOTWRN)
              # (!EPROMCSN & (CNTR : 'H'1) & !VOTRDN & VOTBURSTN)
              # (!EPROMCSN & (CNTR : 'H'7) & !VOTRDN & !VOTBURSTN)
              # (!UARTCSN & (CNTR : 'H'0) & VOTBURSTN)
              # (!TIMERCSN & (CNTR : 'H'0) & VOTBURSTN)

              # (!INTCSN & (CNTR : 'H'0) & VOTBURSTN)
              # (CNTR : 'H'F)
              ));
CYCENDN.AR = LOW;
CYCENDN.SP = LOW;


/* ****************************************************************** */
/*                                                                    */
/*                        READ CLOCK ENABLE                           */
/*                                                                    */
/* THE READ CLOCK ENABLE SIGNAL IS USED BY THE CPU TO STROBE DATA     */
/* OFF THE DATA BUS INTO ITS READ BUFFER.  THIS SIGNAL IS STROBED ONE */
/* TIME FOR SINGLE READS AND FOUR TIMES FOR QUAD READS.  ONLY RAM AND */
/* EPROM MEMORY USE THE QUAD WORD READS.                              */
/* ****************************************************************** */

RDCENN.D = !(RESETN & CYCENDN & !VOTRDN & (

                (!RAMCSN & (
                                  (CNTR : 'H'0)
                    # (!VOTBURSTN & (CNTR : 'H'2))
                    # (!VOTBURSTN & (CNTR : 'H'4))
                    # (!VOTBURSTN & (CNTR : 'H'6))
                    )
              )
              # (!EPROMCSN & (
                                  (CNTR : 'H'1)
                    # (!VOTBURSTN & (CNTR : 'H'3))
                    # (!VOTBURSTN & (CNTR : 'H'5))
                    # (!VOTBURSTN & (CNTR : 'H'7))
                    )
              )
              # (!UARTCSN & (CNTR : 'H'0))
              # (!TIMERCSN & (CNTR : 'H'0))
              # (!INTCSN & (CNTR : 'H'0))
       ));

RDCENN.AR = LOW;
RDCENN.SP = LOW;
```

```
/* ****************************************************************** */
/*                                                                    */
/*                          ACKNOWLEDGE                               */
/*                                                                    */
/* THE ACKNOWLEDGE SIGNAL IS USED BY THE MEMORY SYSTEM TO LET THE     */
/* CPU KNOW THAT IT HAS PROCESSED THE WRITE CYCLE SUFFICIENTLY AND    */
/* THE CPU MAY MOVE ON TO THE NEXT CYCLE. THIS SIGNAL IS GENERATED    */
/* IMPLICITLY ON SINGLE DATUM READS AND NO SOONER THAN FOUR CLOCKS    */
/* BEFORE THE END OF THE LAST READ FOR BURSTS.                        */
/* ****************************************************************** */

ACKN.D = !(RESETN & CYCENDN & (
               (!RAMCSN & !VOTWRN & (CNTR : 'H'0))
             # (!RAMCSN & !VOTRDN & !VOTBURSTN & (CNTR : 'H'3))
             # (!EPROMCSN & !VOTRDN & !VOTBURSTN & (CNTR : 'H'4))
             # (!UARTCSN & !VOTWRN & VOTBURSTN & (CNTR : 'H'0))
             # (!TIMERCSN & !VOTWRN & (CNTR : 'H'0))
             # (!INTCSN & !VOTWRN & (CNTR : 'H'0))
             )
      );

ACKN.AR = LOW;
ACKN.SP = LOW;

/* ****************************************************************** */
/*                                                                    */
/*                          BUS ERROR                                 */
/*                                                                    */
/* THE BUS ERROR SIGNAL IS USED BY THE PROCESSOR TO END A BUS CYCLE   */
/* THAT TRIES TO ACCESS AN ADDRESS THAT IS NOT POPULATED IN THE       */
/* MEMORY SPACE.                                                      */
/* ****************************************************************** */

BUSERRN.D = !(RESETN & CYCENDN & (CNTR : 'H'F));
BUSERRN.AR = LOW;
BUSERRN.SP = LOW;

END;
```

2. MEMORY CONTROL PLD SIMULATION OUTPUT FILE

```
CSIM(WM): CUPL Simulation Program
Version 5.0a Serial# 10000000
Copyright (c) 1983, 1998 Logical Devices, Inc.
CREATED Sat May 27 09:01:57 2000

LISTING FOR SIMULATION FILE: MEMCONT.si

   1: Name     MEMCONT;
   2: PartNo   ATF22V10C-7PC;
   3: Date     5/26/00;
   4: Revision 01;
   5: Designer DAVID SUMMERS;
   6: Company  NPS;
   7: Assembly TMR R3081;
   8: Location U54;
   9: Device   p22v10;
  10:
  11: /**************************************************************/
  12: /* This device generates the memory control and interrupt      */
  13: /* signals required for memory reads and writes.  It also      */
  14: /* generates the voter error interrupt signal.                 */
  15: /**************************************************************/
  16: /* Allowable Target Device Types:      PLD22V10                */
  17: /**************************************************************/
  18:
  19: FIELD CNTR = [COUNT3,COUNT2,COUNT1,COUNT0];
  20:
  21: ORDER: SYSCLK, INTCSN, RAMCSN, TIMERCSN, UARTCSN, EPROMCSN,
      VOTBURSTN, VOTRDN, VOTWRN, CVOTERR, DVOTERR, RESETN, AVOTERR, %2,
      COUNT3..COUNT0, CYCENDN, BUSERRN, ACKN, RDCENN, VOTINTN;
  22:
  23:

_
=================================
            V
         T EO
         IUPT  CD A      CB  V
      SIRMARBVVVVRV  CCCCYU RO
      YNAEROUOOOOEO  OOOOCS DT
      STMRTMRTTTTST  UUUUEEACI
      CCCCCCSRWEEEE  NNNNNRCEN
      LSSSSSTDRRRTR  TTTTDRKNT
      KNNNNNNNNRRNR  3210NNNNN
=================================
0001: C111111110000  LLLLHHHHH
0002: C011111100010  LLLHLHLHH
0003: C011111110010  LLLLHHHHH
0004: C101111100010  LLLHLHLHH
0005: C101111110010  LLLLHHHHH
0006: C110111100010  LLLHLHLHH
0007: C110111110010  LLLLHHHHH
0008: C111011100010  LLLHLHLHH
0009: C111011110010  LLLLHHHHH
0010: C011111010010  LLLHLHHLH
0011: C011111110010  LLLLHHHHH
0012: C101111010010  LLLHLHHLH
0013: C101111110010  LLLLHHHHH
```

```
            V
         T EO
         IUPT  CD A      CB  V
      SIRMARBVVVVRV  CCCCYU RO
      YNAEROUOOOOEO  OOOOCS DT
      STMRTMRTTTTST  UUUUEEACI
      CCCCCCSRWEEEE  NNNNNRCEN
      LSSSSSTDRRRTR  TTTTDRKNT
      KNNNNNNNNRRNR  3210NNNNN
================================
0014: C110111010010  LLLHLHHLH
0015: C110111110010  LLLLHHHHH
0016: C111011010010  LLLHLHHLH
0017: C111011110010  LLLLHHHHH
0018: C111101010010  LLLHHHHHH
0019: C111101010010  LLHLLHHLH
0020: C111101110010  LLLLHHHHH
0021: C101110010010  LLLHHHHLH
0022: C101110010010  LLHLHHHHH
0023: C101110010010  LLHHHHHLH
0024: C101110010010  LHLLHHLHH
0025: C101110010010  LHLHHHHLH
0026: C101110010010  LHHLHHHHH
0027: C101110010010  LHHHLHHLH
0028: C101110010010  LLLLHHHHH
0029: C111100010010  LLLHHHHHH
0030: C111100010010  LLHLHHHLH
0031: C111100010010  LLHHHHHHH
0032: C111100010010  LHLLHHHLH
0033: C111100010010  LHLHHHLHH
0034: C111100010010  LHHLHHHLH
0035: C111100010010  LHHHHHHHH
0036: C111100010010  HLLLLHHLH
0037: C111100010010  LLLLHHHHH
0038: C111111010010  LLLHHHHHH
0039: C111111010010  LLHLHHHHH
0040: C111111010010  LLHHHHHHH
0041: C111111010010  LHLLHHHHH
0042: C111111010010  LHLHHHHHH
0043: C111111010010  LHHLHHHHH
0044: C111111010010  LHHHHHHHH
0045: C111111010010  HLLLHHHHH
0046: C111111010010  HLLHHHHHH
0047: C111111010010  HLHLHHHHH
0048: C111111010010  HLHHHHHHH
0049: C111111010010  HHLLHHHHH
0050: C111111010010  HHLHHHHHH
0051: C111111010010  HHHLHHHHH
0052: C111111010010  HHHHHHHHH
0053: C111111010010  LLLLLLHHH
0054: C111111010010  LLLLHHHHH
0055: C111111110000  LLLLHHHHH
0056: C111111111010  LLLLHHHHL
0057: C011111100010  LLLHLHLHH
0058: C011111110010  LLLLHHHHH
0059: C111111110110  LLLLHHHHL
0060: C011111010010  LLLHLHHLH
0061: C011111110010  LLLLHHHHH
0062: C111111110011  LLLLHHHHL
0063: C011111110010  LLLLLHHHH
```

3. MEMORY ENABLE PLD PROGRAM FILE

```
Name     MemEn;
PartNo   ATF22V10C-7PC;
Date     4/30/00;
Revision 01;
Designer David Summers;
Company  NPS;
Assembly TMR R3081;
Location U55;
Device   p22v10;

/* ***********************      INPUT PINS      *********************** */

PIN   1 = SYSCLK;                   /*  SYSCLK FR CPUA                  */
PIN   2 = VOTRDN;                   /*  VOTED READ                      */
PIN   3 = VOTWRN;                   /*  VOTED WRITE                     */
PIN   4 = CYCENDN;                  /*  CYCLE END SIGNAL                */
PIN   6 = VOTBE0N;                  /*  VOTED BYTE ENABLE 0             */
PIN   7 = VOTBE1N;                  /*  VOTED BYTE ENABLE 1             */
PIN   8 = VOTBE2N;                  /*  VOTED BYTE ENABLE 2             */
PIN   9 = VOTBE3N;                  /*  VOTED BYTE ENABLE 3             */
PIN  13 = SYS_RESETN;               /*  SYSTEM RESET                    */

/* ***********************     OUTPUT PINS      *********************** */

PIN  14 = RDDATAENN;         /* READ DATA ENABLE FOR XCEIVER ENABLE    */
PIN  15 = WRDATAEN;          /* WRITE DATA ENABLE FOR XCEIVER ENABLE   */
PIN  16 = RDENN;             /* READ ENABLE                            */
PIN  17 = WRENDN;            /* WRITE ENABLE FOR BYTE 3                */
PIN  18 = WRENCN;            /* WRITE ENABLE FOR BYTE 2                */
PIN  19 = WRENBN;            /* WRITE ENABLE FOR BYTE 1                */
PIN  20 = WRENAN;            /* WRITE ENABLE FOR BYTE 0                */
PIN  22 = RESETPOS;          /* SYNCHRONOUS SYSTEM RESET FOR UART      */
PIN  23 = RESETN;            /* SYNCHRONOUS SYSTEM RESET               */

/* ******************    CONSTANT DEFINITIONS     ****************** */

$DEFINE LOW  'B'0
$DEFINE HIGH 'B'1


/* *********************** LOGIC EQUATIONS  *********************** */

/* ***************************************************************** */
/*                                                                   */
/*                         READ ENABLE STROBE                        */
/*                                                                   */
/* THE READ ENABLE STROBE IS USED TO ACTIVATE THE OUTPUT ENABLES ON  */
/* THE MEMORY AND I/O DEVICES.  SINCE ALL READS ARE DONE 32 BITS WIDE */
/* THERE IS NO NEED TO DIFFERENTIATE BETWEEN THE BYTES ON THE BUS.   */
/* THE SIGNAL IS INITIATED BY THE VOTED READ AND DISABLED BY THE     */
/* CYCLE END SIGNAL.                                                 */
/* ***************************************************************** */

RDENN.D =!(SYS_RESETN & !VOTRDN & CYCENDN);

RDENN.AR = LOW;
RDENN.SP = LOW;
```

```
/* ****************************************************************** */
/*                                                                    */
/*                       WRITE ENABLE STROBES                         */
/*                                                                    */
/* THE WRITE ENABLE STROBES DETERMINE WHICH BYTES OF THE 32 BIT DATA  */
/* BUS WILL BE INVOLVED IN THE WRITE CYCLE, SINCE THE R3081 CAN       */
/* PERFORM PARTIAL WORD WRITES.  THE WRITE ENABLE STROBES CORRESPOND  */
/* TO THE BYTES ON THE DATA BUS AS FOLLOWS:                           */
/*                 WRENAN  ->  DATA(7..0)                             */
/*                 WRENBN  ->  DATA(15..8)                            */
/*                 WRENCN  ->  DATA(23..16)                           */
/*                 WRENDN  ->  DATA(31..24)                           */
/* THE !VOTWRN AND !VOTBE_N SIGNALS INITIATES THE STROBE AND THE      */
/* CYCENDN SIGNAL DEACTIVATES THEM.  ONE TO ALL OF THE STROBES CAN BE */
/* ACTIVE DURING THE WRITE CYCLE.                                     */
/* ****************************************************************** */

WRENAN.D = !(SYS_RESETN & (!VOTWRN & !VOTBE0N & CYCENDN));
WRENBN.D = !(SYS_RESETN & (!VOTWRN & !VOTBE1N & CYCENDN));
WRENCN.D = !(SYS_RESETN & (!VOTWRN & !VOTBE2N & CYCENDN));
WRENDN.D = !(SYS_RESETN & (!VOTWRN & !VOTBE3N & CYCENDN));

WRENAN.AR = LOW;
WRENBN.AR = LOW;
WRENCN.AR = LOW;
WRENDN.AR = LOW;

WRENAN.SP = LOW;
WRENBN.SP = LOW;
WRENCN.SP = LOW;
WRENDN.SP = LOW;


/* ****************************************************************** */
/*                                                                    */
/*          READ DATA ENABLE AND WRITE DATA ENABLE STROBES            */
/*                                                                    */
/* THE READ DATA ENABLE AND WRITE DATA ENABLE STROBES DETERMINE THE   */
/* DIRECTION DATA TRAVELS THROUGH THE DATA VOTER FPGA AND THE         */
/* 74AHCT623 BUS TRANSCEIVER.  THE READ AND WRITE SIGNALS COME ON TO  */
/* EARLY IN THE CYCLE AND STAY ON TOO LATE IN THE CYCLE TO BE USED.   */
/* THESE SIGNALS PREVENT BUS CONTENTION WHEN DEVICES WITH LONG TURN   */
/* OFF TIMES DRIVE THE BUS JUST PRIOR TO IT BEING TURNED AROUND AND   */
/* DRIVEN BY THE PROCESSOR.                                           */
/* ****************************************************************** */

WRDATAEN.D = SYS_RESETN & !VOTWRN & CYCENDN;
RDDATAENN.D = !(SYS_RESETN & !VOTRDN & CYCENDN);

WRDATAEN.AR = LOW;
RDDATAENN.AR = LOW;

WRDATAEN.SP = LOW;
RDDATAENN.SP = LOW;
```

```
/* ******************************************************************* */
/*                                                                     */
/*                        RESETPOS AND RESETN                          */
/*                                                                     */
/* THESE TWO SIGNALS PROVIDE SYNCHRONOUS POSITIVE AND NEGATIVE LOGIC  */
/* RESETS FOR THE SYSTEM.                                              */
/* ******************************************************************* */

RESETPOS.D  = !SYS_RESETN;
RESETN.D = SYS_RESETN;

RESETPOS.AR = LOW;
RESETN.AR = LOW;

RESETPOS.SP = LOW;
RESETN.SP = LOW;


END;
```

4. MEMORY ENABLE PLD SIMULATION FILE

```
CSIM(WM): CUPL Simulation Program
Version 5.0a Serial#
Copyright (c) 1983, 1998 Logical Devices, Inc.
CREATED Fri May 26 13:19:41 2000

LISTING FOR SIMULATION FILE: MEMEN.si

Name     MEMEN;
PartNo   ATF22V10C-7PC;
Date     5/26/00;
Revision 01;
Designer DAVID SUMMERS;
Company  NPS;
Assembly TMR R3081;
Location U55;
Device   p22v10;

/******************************************************************/
/* This device generates the memory write byte enable signals    */
/* and positive and negative logic synchronous reset signals.    */
/* It also generates the read and write data enable strobes that*/
/* control the tri-state output buffers on the bus transceivers */
/* and FPGAs.                                                    */
/******************************************************************/
/* Allowable Target Device Types:       PLD22V10                 */
/******************************************************************/

ORDER: SYSCLK, SYS_RESETN, VOTRDN, VOTWRN, CYCENDN, %1, VOTBE0N,
VOTBE1N, VOTBE2N, VOTBE3N, %1, WRDATAEN, RDDATAENN, %1, RDENN, %1,
WRENDN, WRENCN, WRENBN, WRENAN, %1, RESETPOS, RESETN;


_
==============================
       S
       Y          R
       S         WD        R
       _  C VVVV RD        E
      SRVVY 0000 DA   WWWW SR
      YEOOC TTTT AT R RRRR EE
      SSTTE BBBB TA D EEEE TS
      CERWN EEEE AE E NNNN PE
      LTDRD 0123 EN N DCBA OT
      KNNNN NNNN NN N NNNN SN
==============================
0001: C0111 1111 LH H HHHH HL
0002: C1001 1111 HL L HHHH LH
0003: C1000 1111 LH H HHHH LH
0004: C1110 1111 LH H HHHH LH
0005: C1111 1111 LH H HHHH LH
0006: C1101 0000 HH H LLLL LH
0007: C1100 0000 LH H HHHH LH
0008: C1101 0000 HH H LLLL LH
0009: C1101 1111 HH H HHHH LH
0010: C0101 1111 LH H HHHH HL
```

APPENDIX C. XILINX FOUNDATION DESIGNS

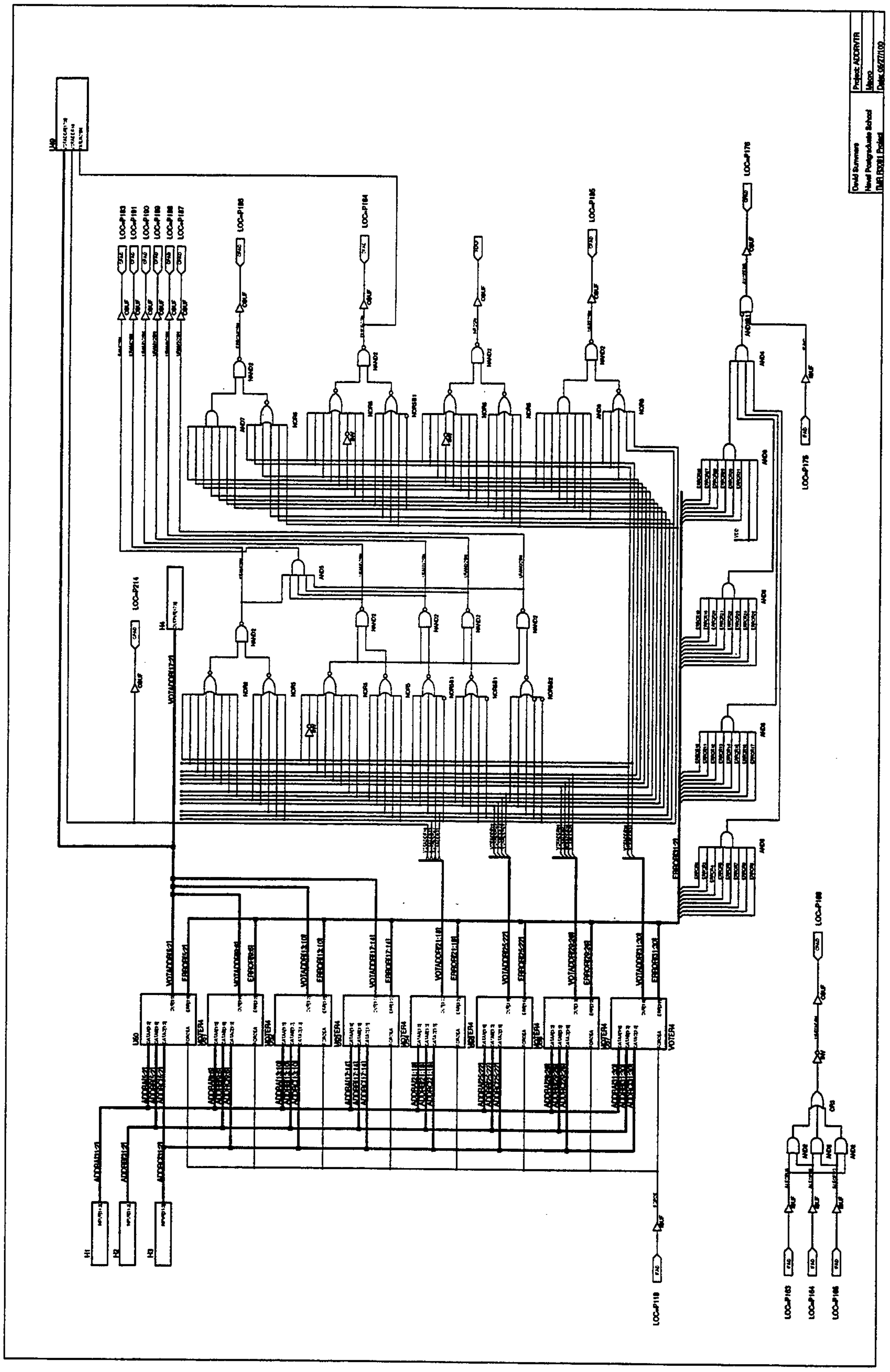
Just as with the PLDs, the FPGAs have to be programmed. Although the WinCUPL software had provisions for programming Xilinx FPGAs, it required the files to be written in HDL or Verilog. An alternative offered by the Xilinx Foundation software is schematic programming of the FPGA. The designer enters the design in a schematic format and the software converts the file to a netlist, which is compiled into the appropriate format for programming the FPGA. Section A of this appendix contains the Foundation Schematics for the design of the Address Voter FPGA and Section B contains the schematics for the Data and Control Voter FPGA. Each of these FPGA designs were presented in Chapter IV.

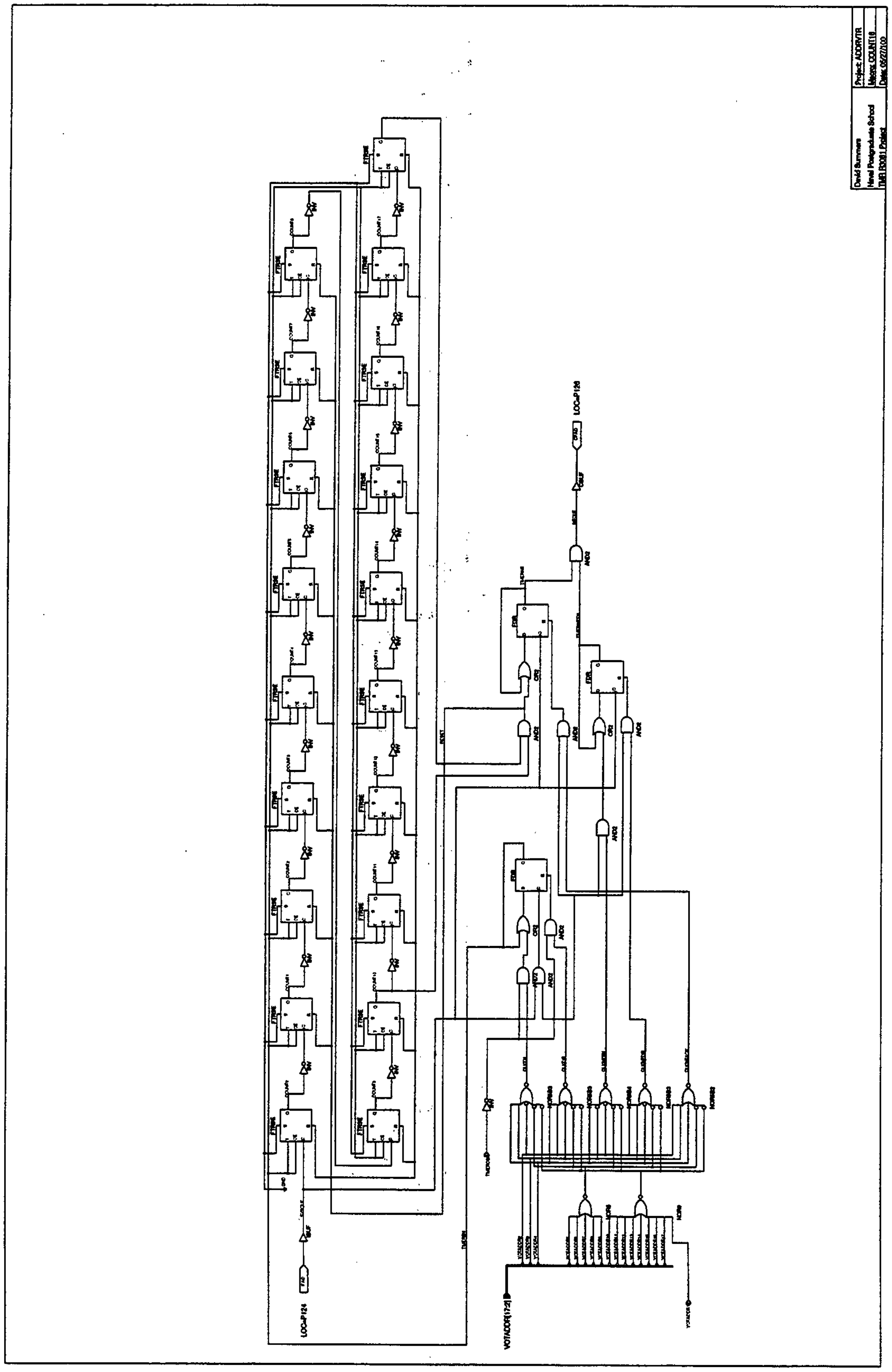
1. ADDRESS VOTER FPGA

The Address Voter FPGA is responsible for performing a majority vote on the three address busses, decoding the upper thirteen bits of the address bus into chip select signals, and providing the system timer interrupt. The design of this FPGA is presented in the following figures. Table C.1 lists the figures presented in this section.

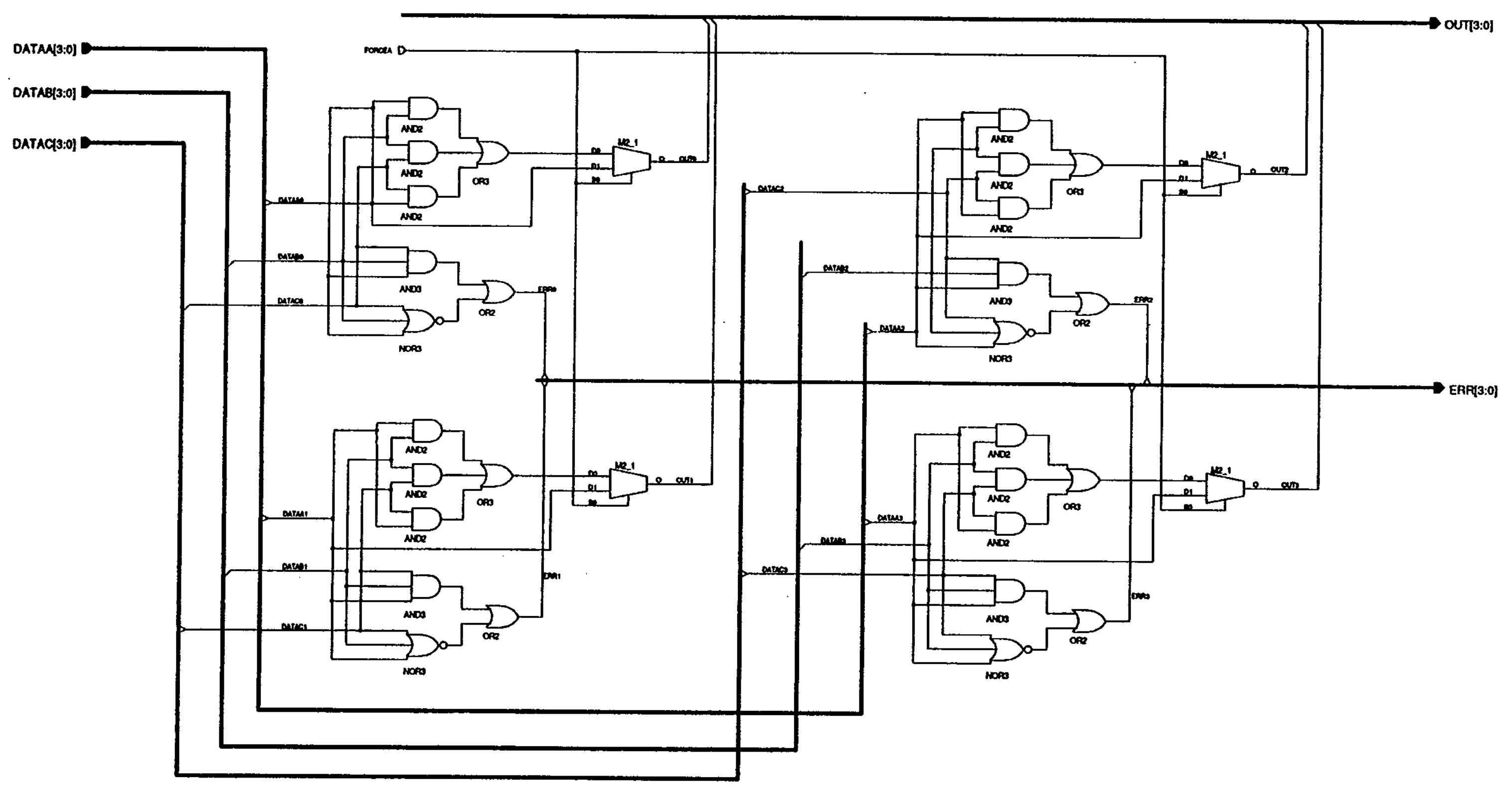
Figure Number and Description	Page Number
Figure C.1. Address Voter FPGA Top Level Schematic	146
Figure C.2. 18-Bit Counter	147
Figure C.3. 4-Bit Wide 3-Bit Majority Voter	148
Figure C.4. CPU A Address Bus Input Specification	149
Figure C.5. CPU B Address Bus Input Specification	150
Figure C.6. CPU C Address Bus Input Specification	151
Figure C.7. Voted Address Bus Output Specification	152

Table C.1. Address Voter Figures By Page Number

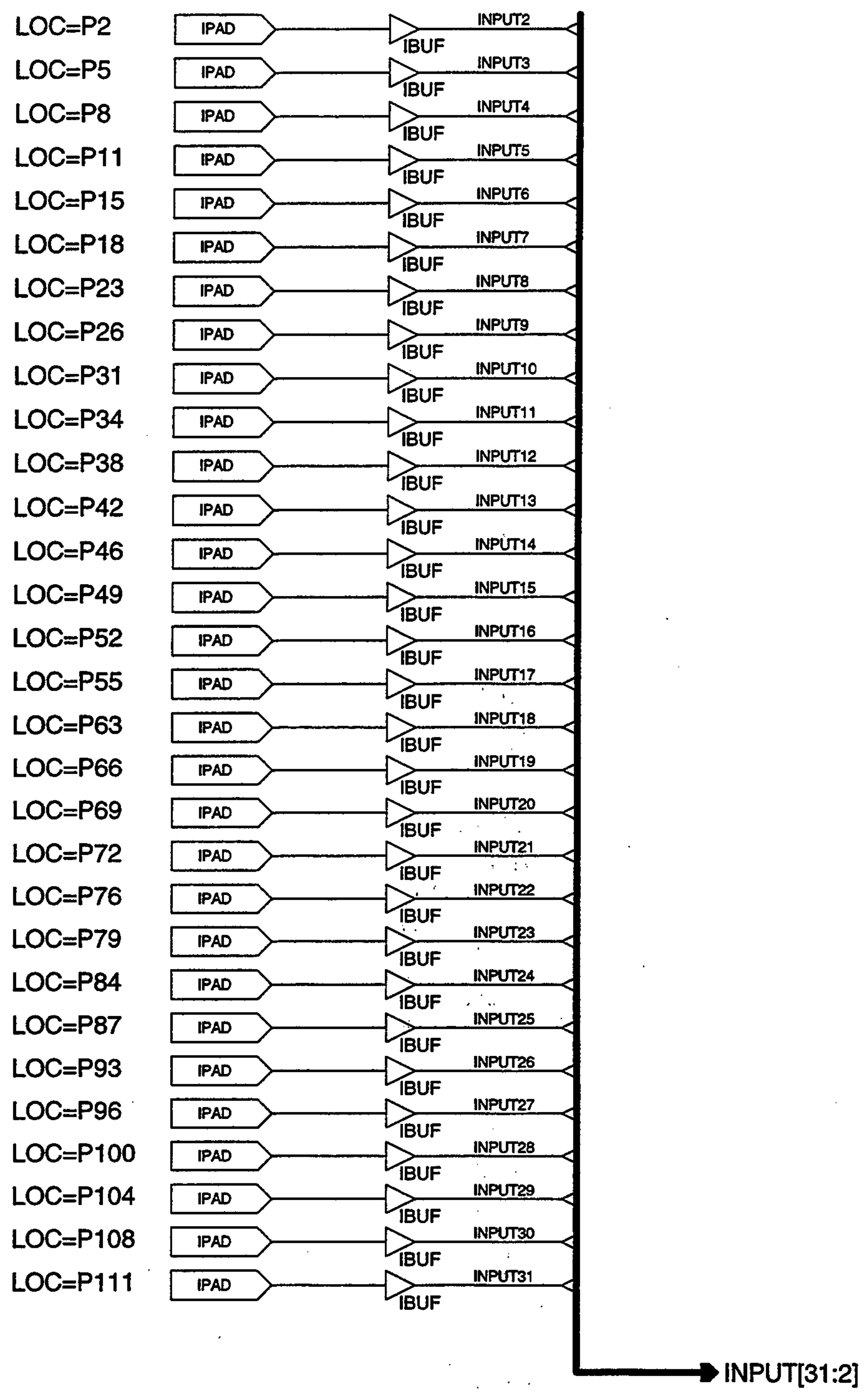

David Summers
Naval Postgraduate School
TMR RX281 Project
Project: ADDRVTR
Macro
Date: 06/27/100
VOTER4
NAND2
NOR8
AND8
AND4
OBUF
IBUF
LOC=P118
LOC=P163
LOC=P164
LOC=P166
LOC=P175
LOC=P176
LOC=P185
LOC=P184
LOC=P214



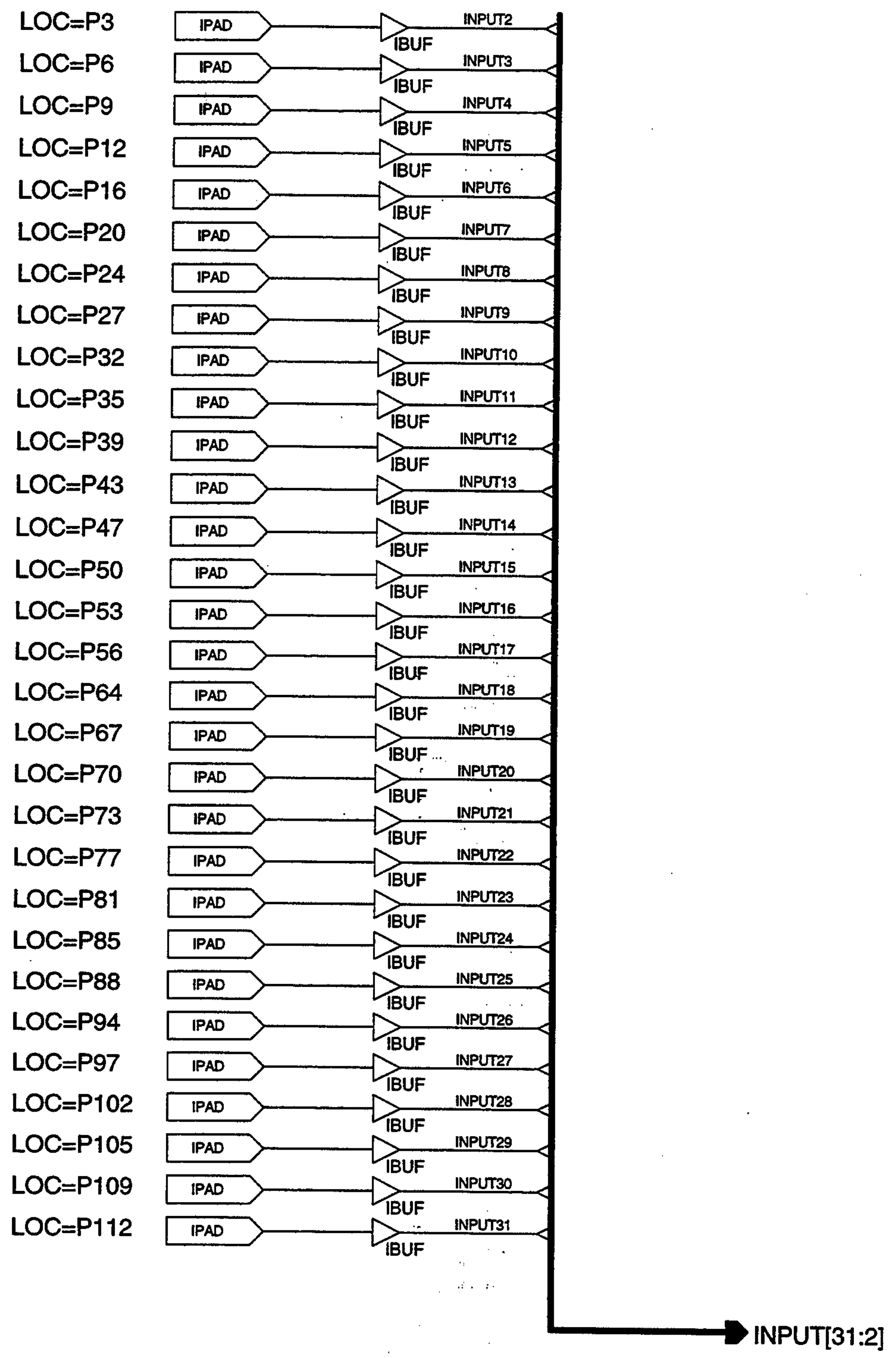
David Summers
Naval Postgraduate School
TMR RO061 Project
Project: ADDRVTR
Macro: COUNT18
Date: 05/27/00
LOC=P124
LOC=P126
VOTADDR[17:2]
FTCE
FDR
AND2
OR2
BUF
INV
NOR8



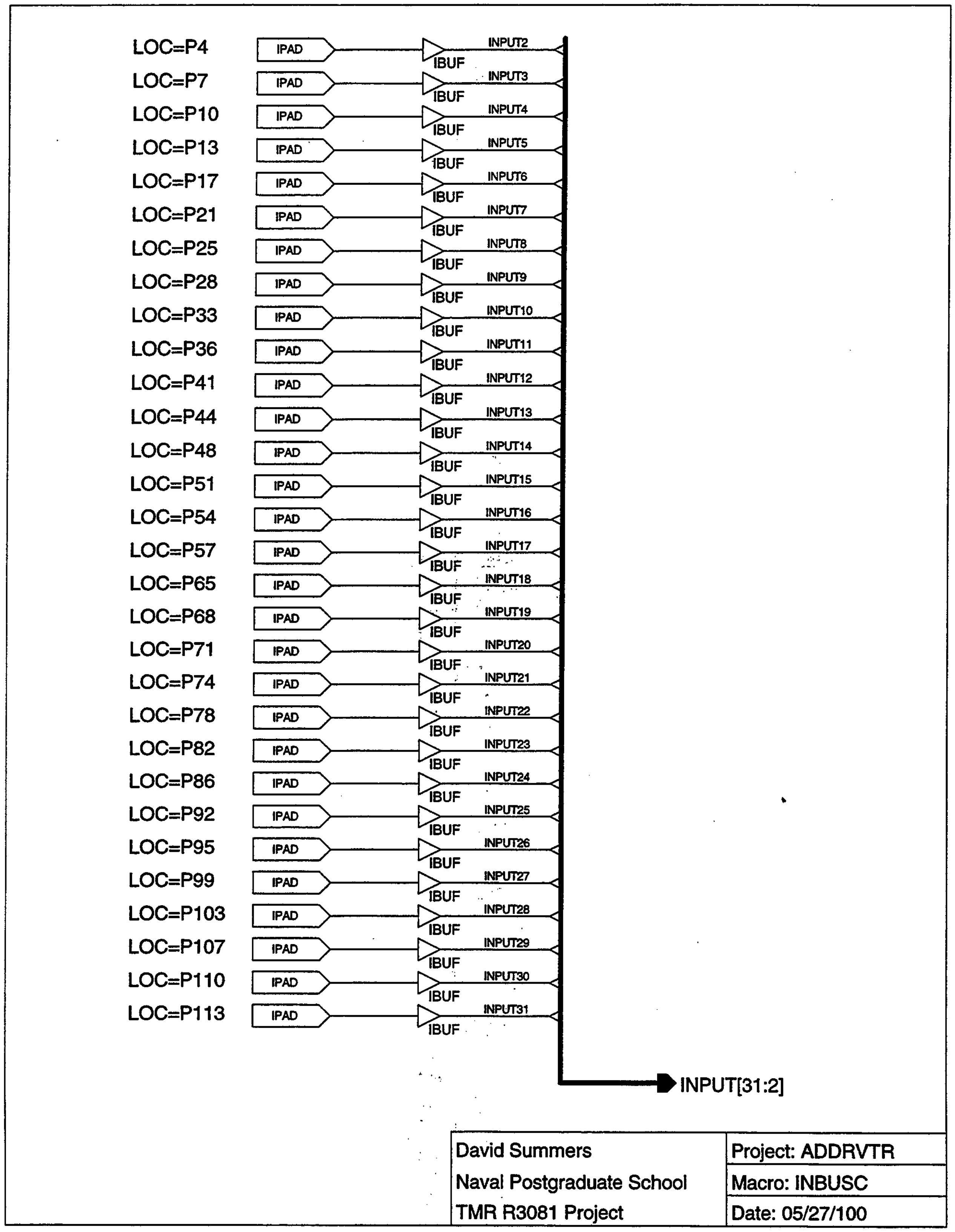
David Summers	Project: ADDRVTR
Naval Postgraduate School	Macro: VOTER4
TMR R3081 Project	Date: 05/27/100



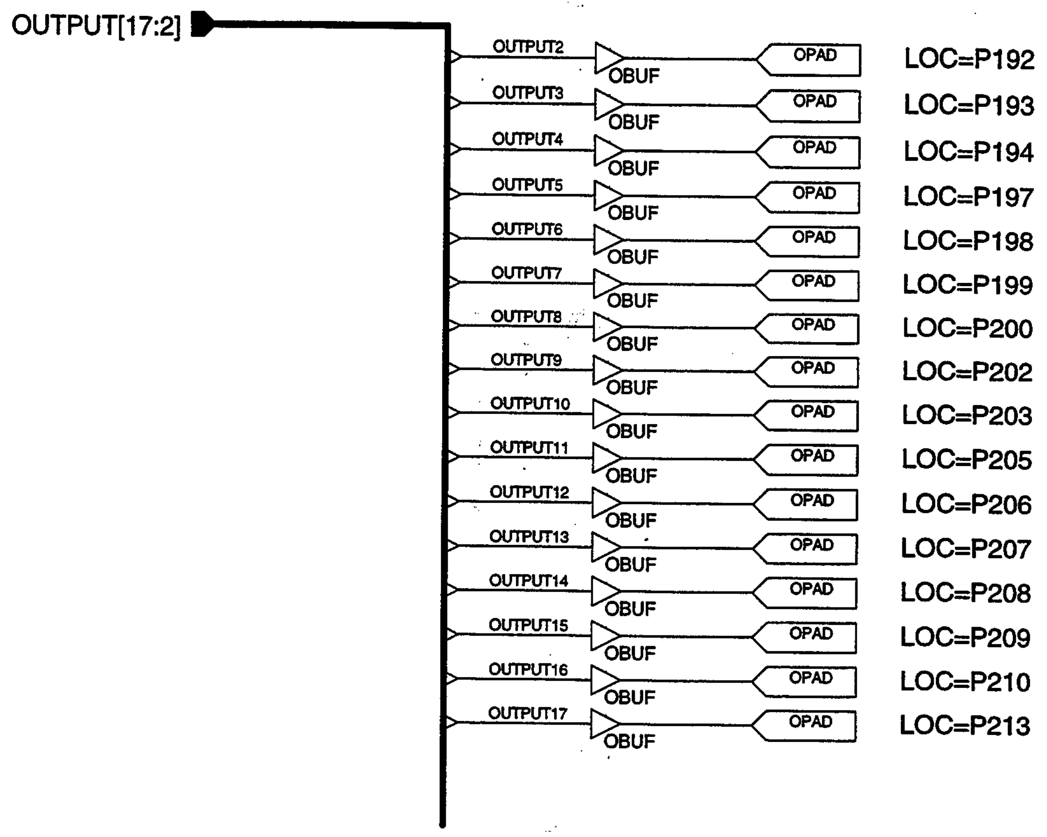
David Summers	Project: ADDRVTR
Naval Postgraduate School	Macro: INBUSA
TMR R3081 Project	Date: 05/27/100



David Summers	Project: ADDRVTR
Naval Postgraduate School	Macro: INBUSB
TMR R3081 Project	Date: 05/27/100



David Summers	Project: ADDRVTR
Naval Postgraduate School	Macro: INBUSC
TMR R3081 Project	Date: 05/27/100



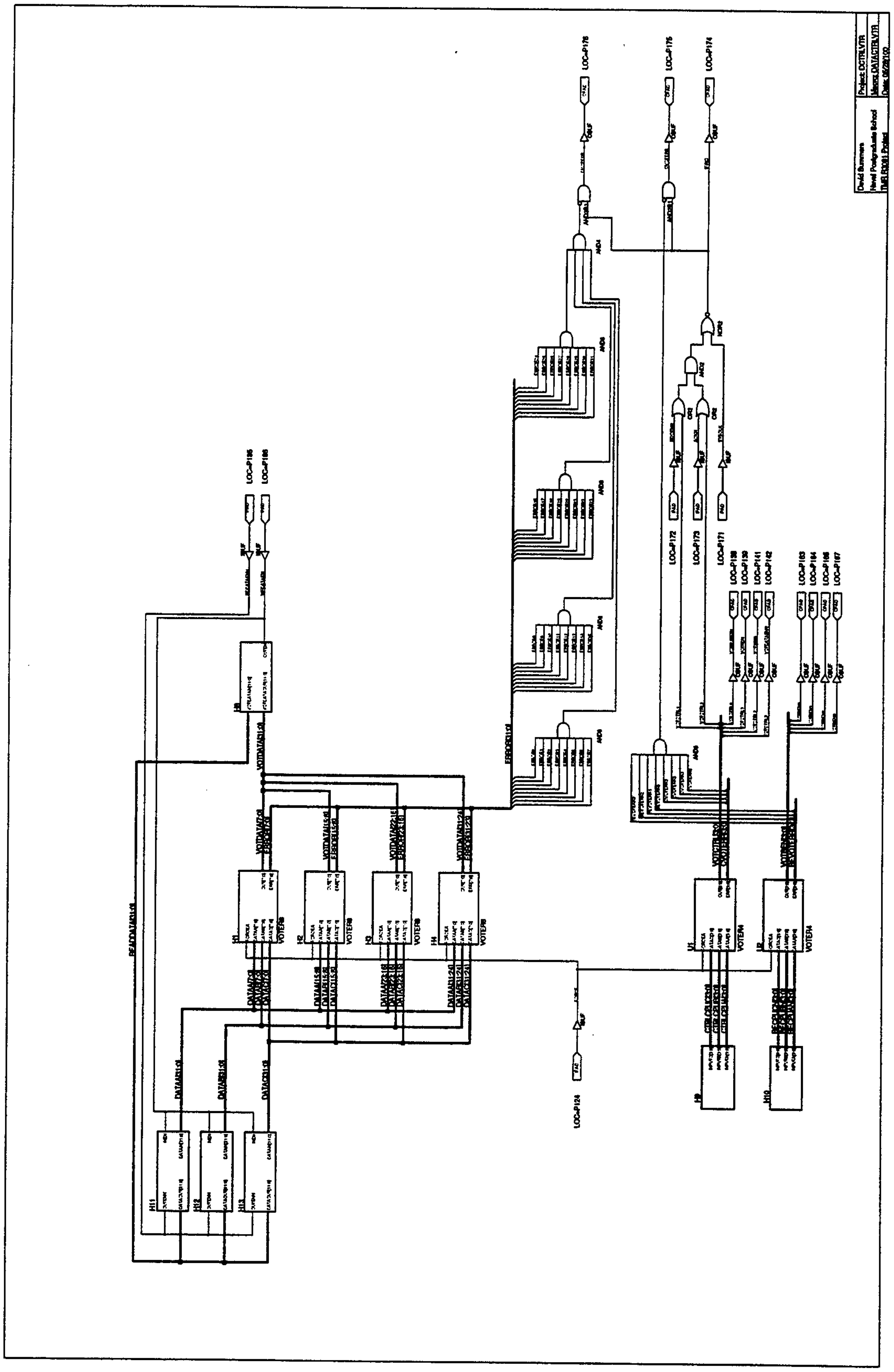
David Summers	Project: ADDRVTR
Naval Postgraduate School	Macro: OUTVOT
TMR R3081 Project	Date: 05/27/100

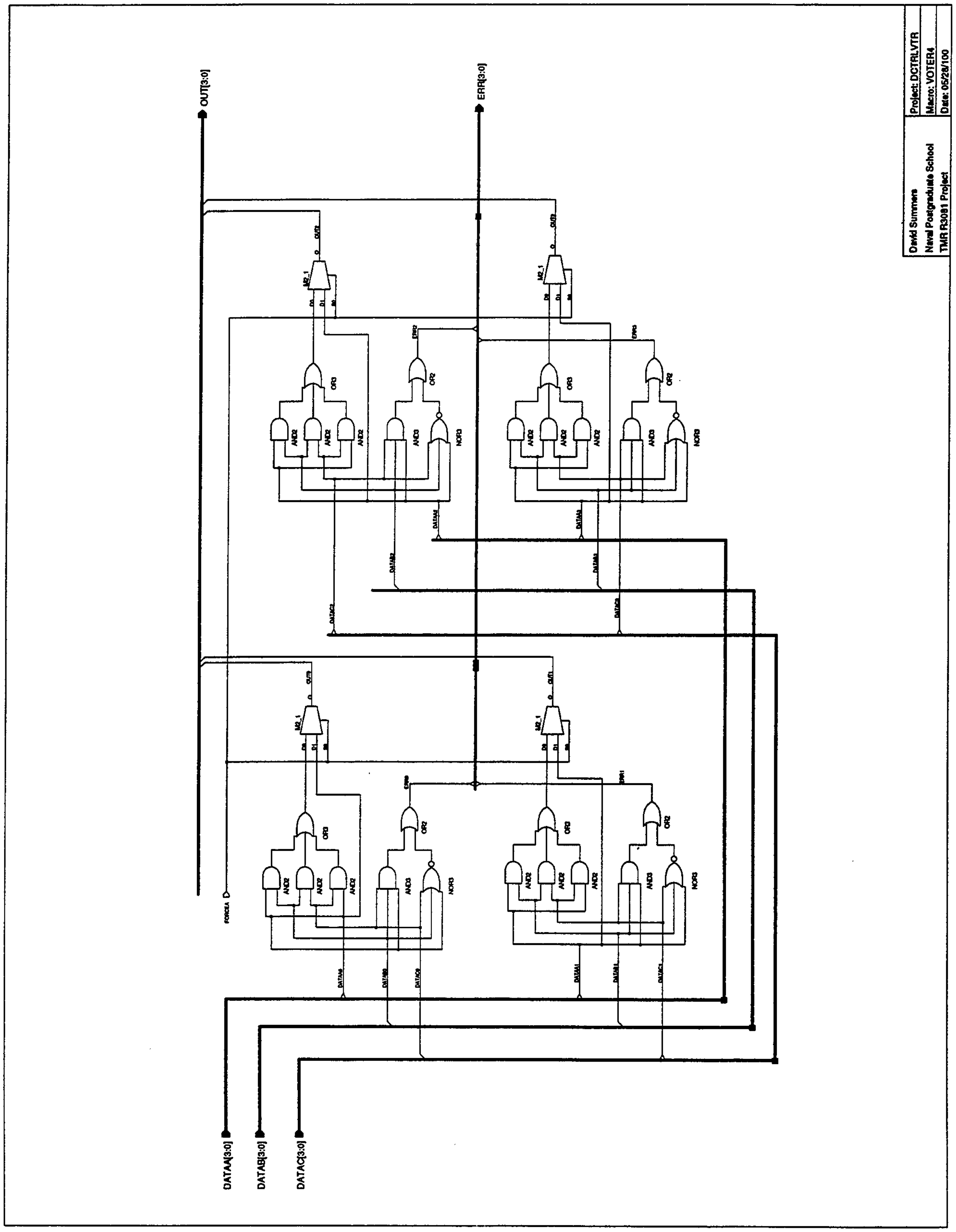
2. DATA AND CONTROL VOTER FPGA

The Data and Control Voter FPGA performs a three bit majority vote on the Data and Control Busses of the three microprocessors. The Data Bus is a 32-bit wide bus and the Control Bus is an 8-bit wide bus. Because the Data Bus is bidirectional, the FPGA must be able to vote processor data to memory on writes and pass memory data back to all three processors on reads. This FPGA incorporates both 4-bit wide voters and 8-bit wide voters in its design. The design of this FPGA is presented in the following figures. Table C.2 lists the figures presented in this section.

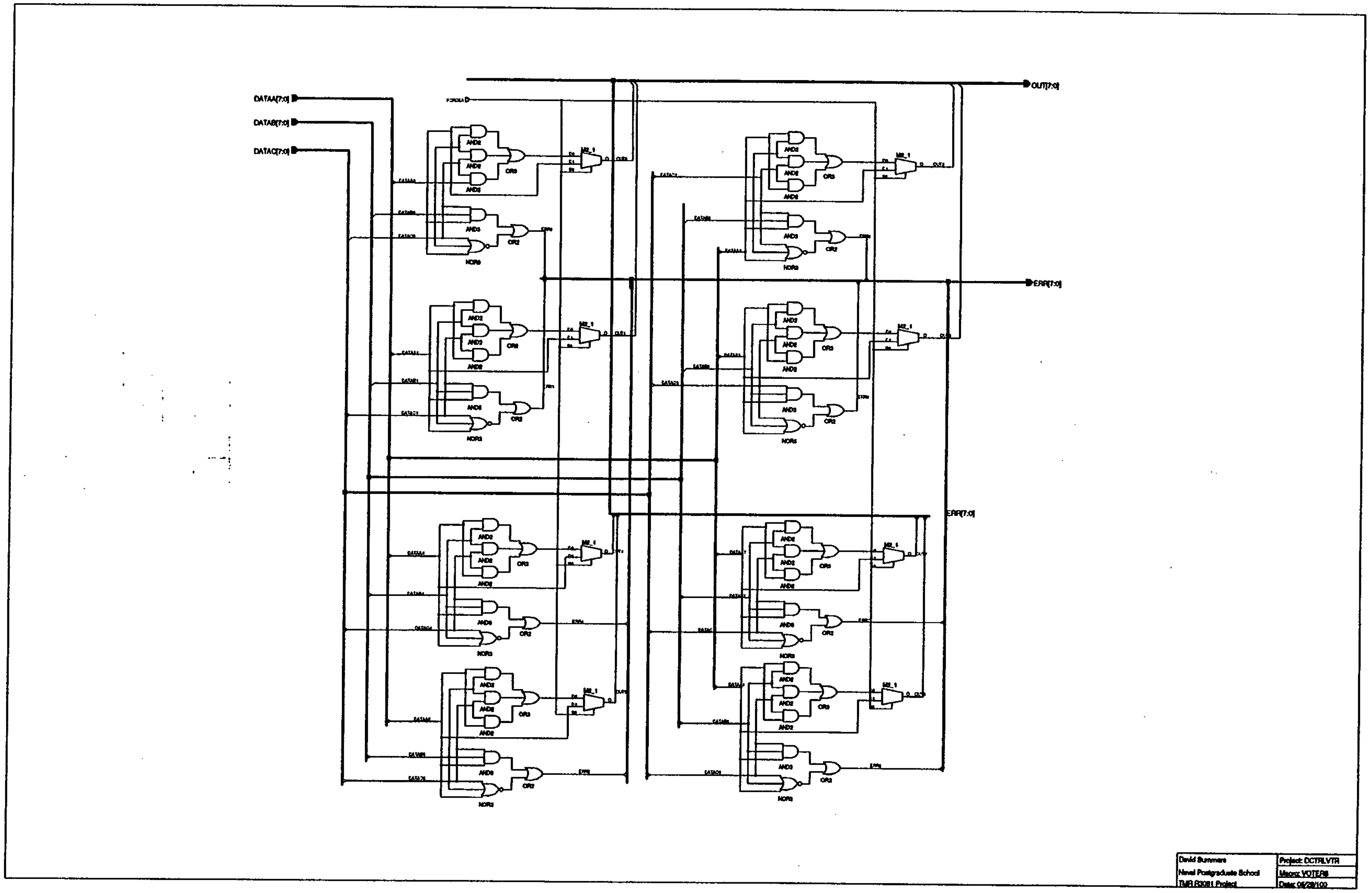
Figure Number and Description	Page Number
Figure C.8. Data and Control Voter FPGA Top Level Schematic	154
Figure C.9. 4-Bit Wide 3-Bit Majority Voter	155
Figure C.10. 8-Bit Wide 3-Bit Majority Voter	156
Figure C.11. Control Bus Input Specification	157
Figure C.12. Byte Enable Bus Input Specification	158
Figure C.13. CPU A Data Bus Input Specification	159
Figure C.14. CPU B Data Bus Input Specification	160
Figure C.15. CPU C Data Bus Input Specification	161
Figure C.16. Voted Data Bus Output Specification	162

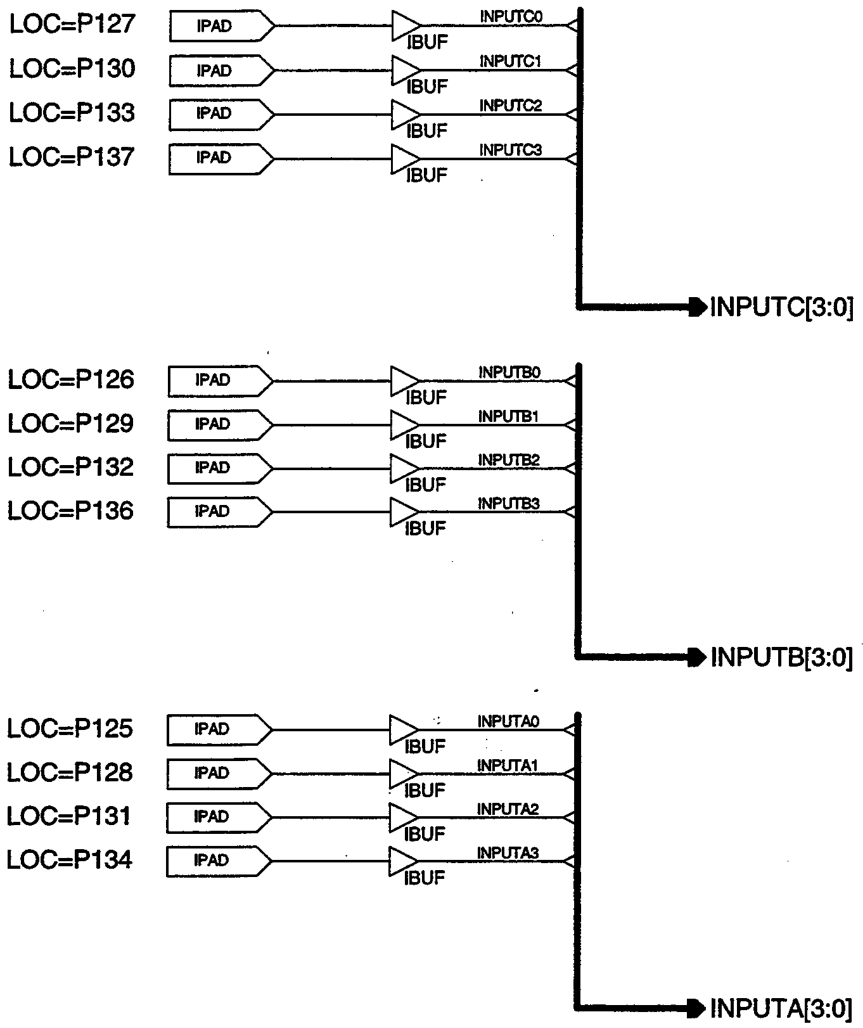
Table C.2. Data and Control Voter Figures By Page Number


David Summers
Naval Postgraduate School
TMR DC-B1 Project
Project: DCTRLVTR
Macro: DATACTRLVTR
Date: 05/28/100
LOC=P176
LOC=P175
LOC=P174
LOC=P185
LOC=P186
LOC=P172
LOC=P173
LOC=P171
LOC=P138
LOC=P139
LOC=P141
LOC=P142
LOC=P163
LOC=P164
LOC=P166
LOC=P167
LOC=P124
READDATA[31:0]
VOTDATA[31:0]
ERROR[31:0]
DATAA[31:0]
DATAB[31:0]
DATAC[31:0]
VOTER8
VOTER4
AND4
AND8
NOR2
OR2
AND2
OBUF
IBUF
H1
H2
H3
H4
H8
H9
H10
H11
H12
H13
U1
U2

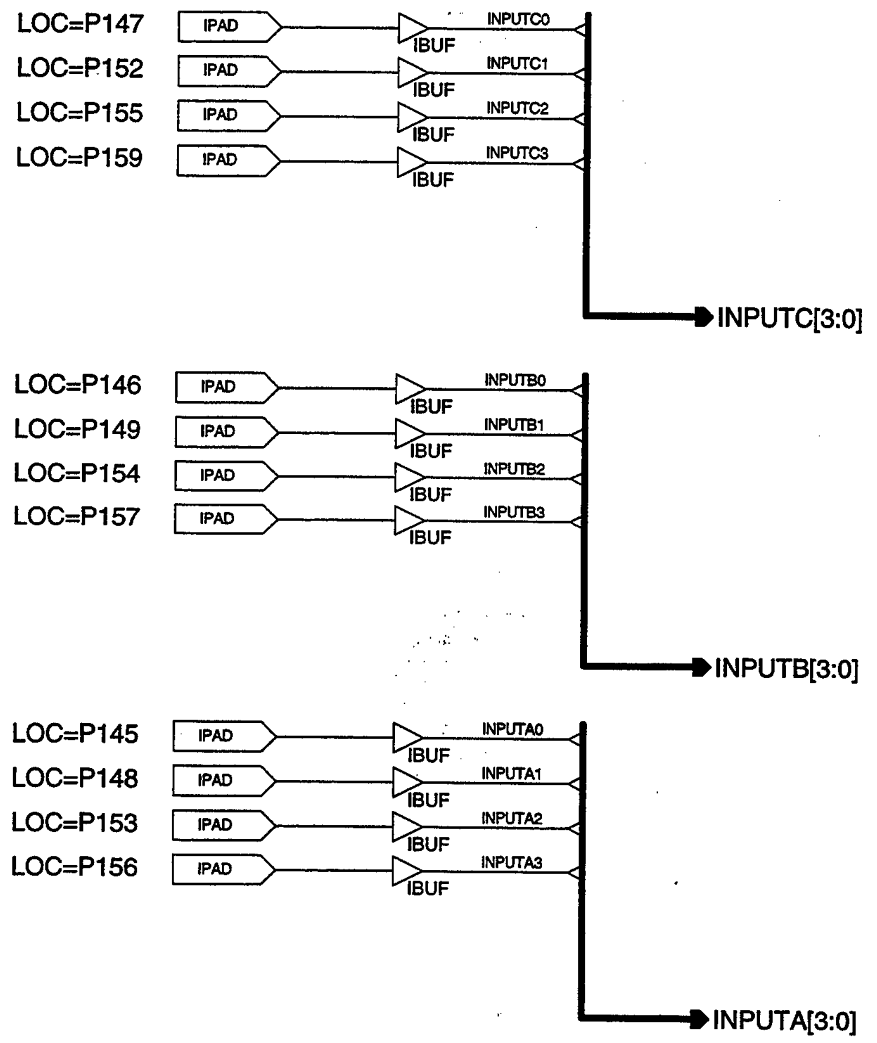


OUT[3:0]
ERR[3:0]
DATAA[3:0]
DATAB[3:0]
DATAC[3:0]
FORCEA
M2_1
D0
D1
S0
O
OUT0
OUT1
OUT2
OUT3
ERR0
ERR1
ERR2
ERR3
AND2
AND3
OR2
OR3
NOR3
DATAA0
DATAB0
DATAC0
DATAA1
DATAB1
DATAC1
DATAA2
DATAB2
DATAC2
DATAA3
DATAB3
DATAC3
David Summers
Naval Postgraduate School
TMR R3081 Project
Project: DCTRLVTR
Macro: VOTER4
Date: 05/28/100

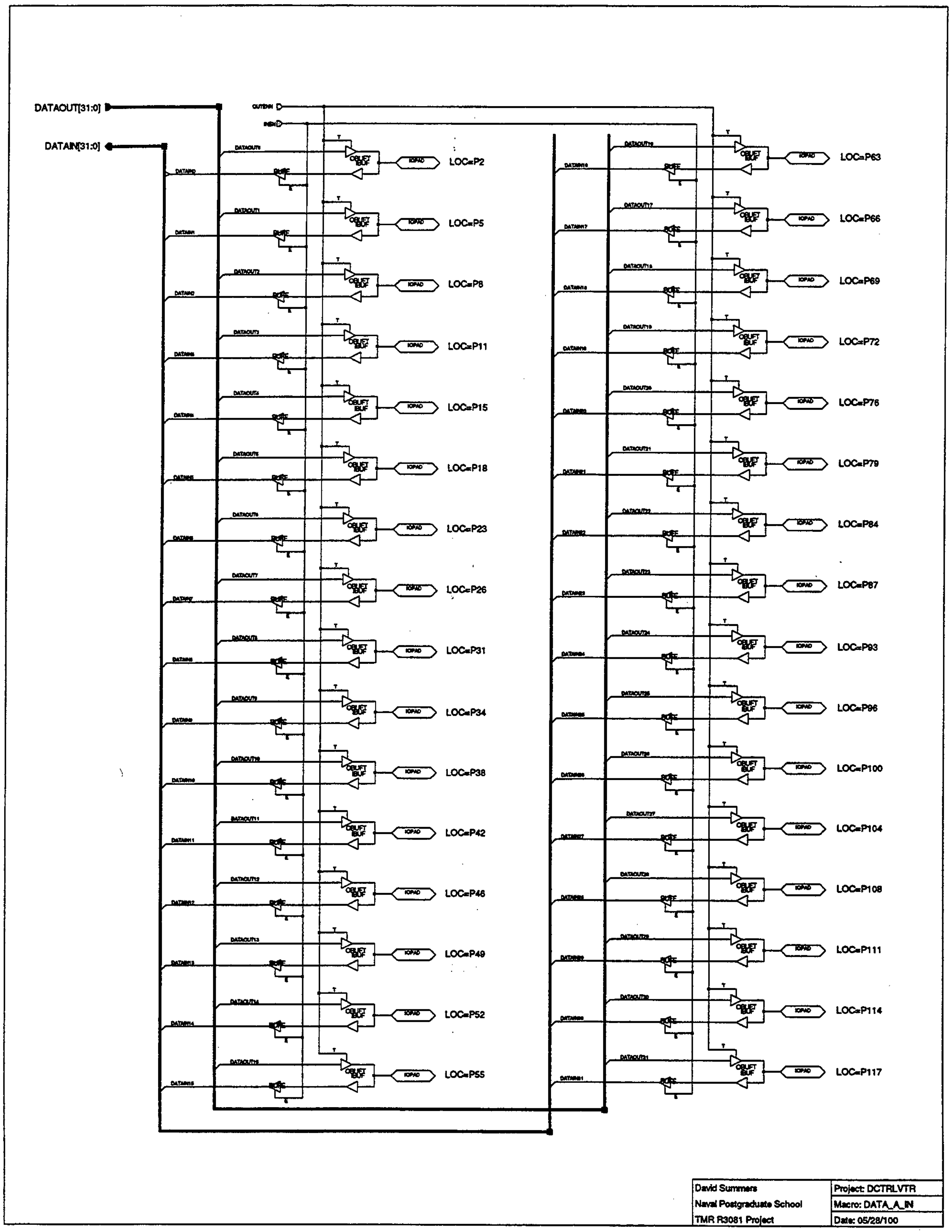

DATAA[7:0]
DATAB[7:0]
DATAC[7:0]
OUT[7:0]
ERR[7:0]
AND2
AND3
OR3
OR2
NOR3
M2_1
David Summers
Naval Postgraduate School
TMR R3081 Project
Project: DCTRLVTR
Macro: VOTER8
Date: 06/28/100



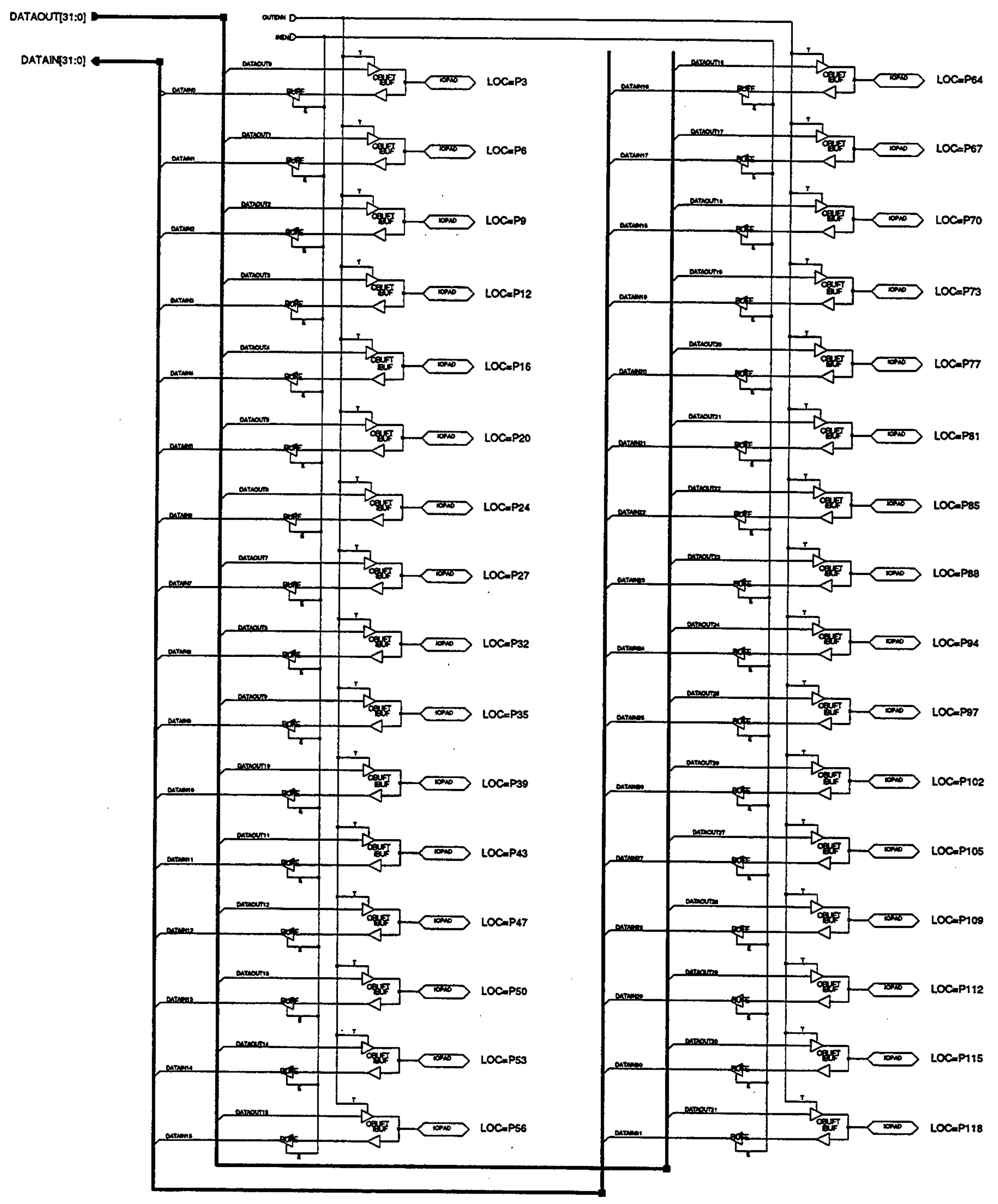
David Summers	Project: DCTRLVTR
Naval Postgraduate School	Macro: CTRL_IN
TMR R3081 Project	Date: 05/28/100



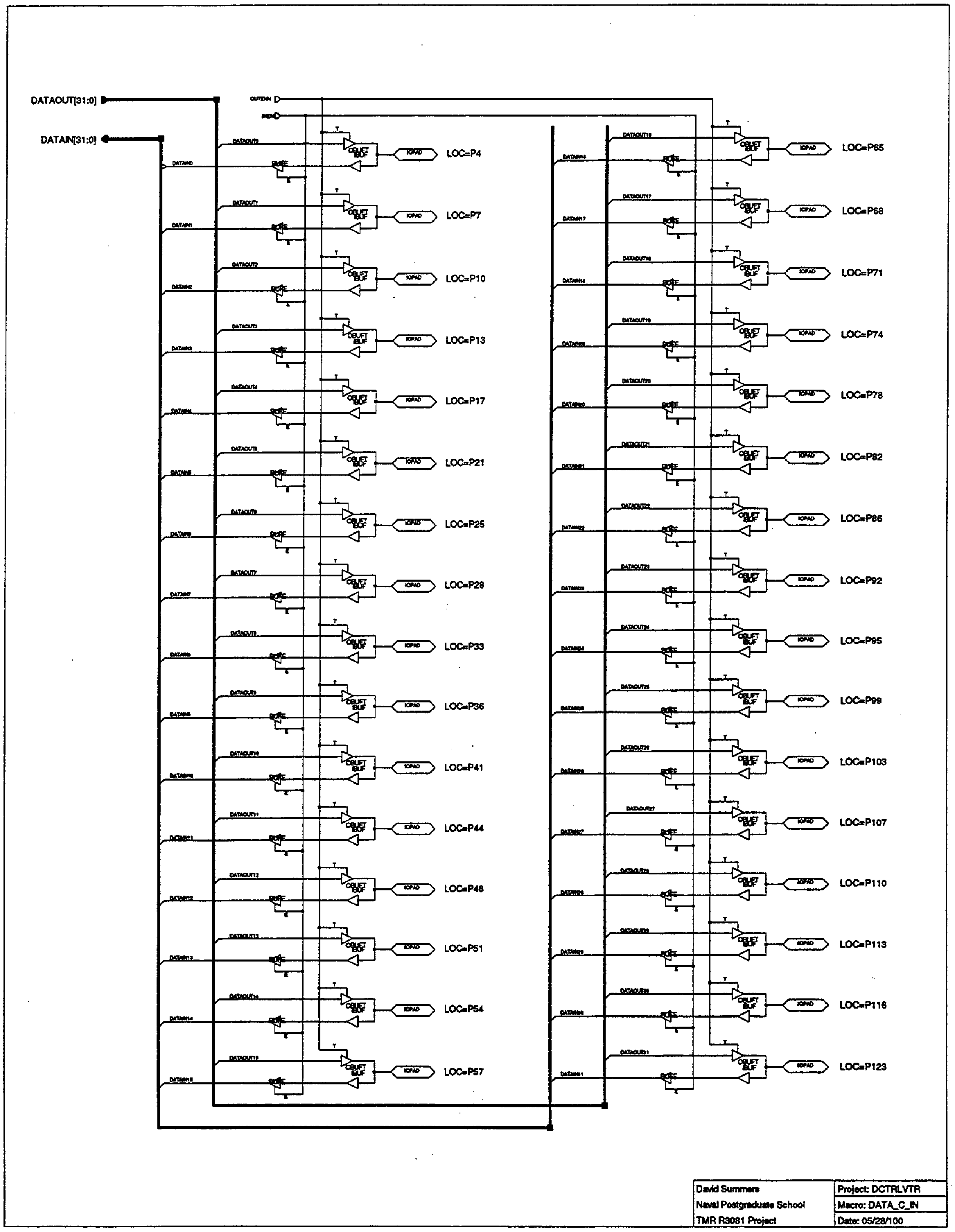
David Summers	Project: DCTRLVTR
Naval Postgraduate School	Macro: BEN_IN
TMR R3081 Project	Date: 05/28/100

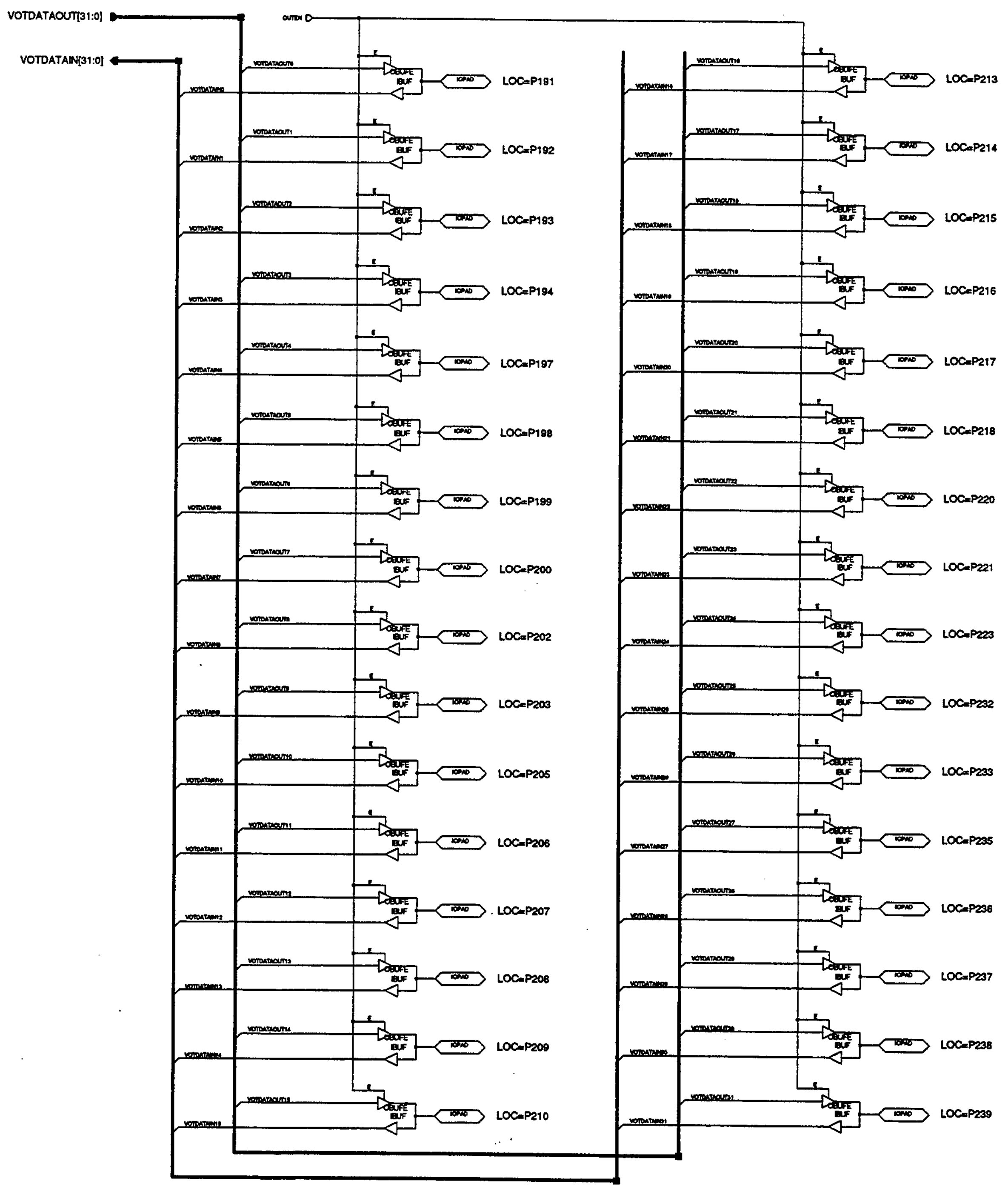


David Summers	Project: DCTRLVTR
Naval Postgraduate School	Macro: DATA_A_IN
TMR R3081 Project	Date: 05/28/100



David Summers	Project: DCTRLVTR
Naval Postgraduate School	Macro: DATA_B_IN
TMR R3081 Project	Date: 05/28/100


DATAOUT[31:0]
DATAIN[31:0]
OUTEN
INEN
LOC=P4
LOC=P7
LOC=P10
LOC=P13
LOC=P17
LOC=P21
LOC=P25
LOC=P28
LOC=P33
LOC=P36
LOC=P41
LOC=P44
LOC=P48
LOC=P51
LOC=P54
LOC=P57
LOC=P65
LOC=P68
LOC=P71
LOC=P74
LOC=P78
LOC=P82
LOC=P86
LOC=P92
LOC=P95
LOC=P99
LOC=P103
LOC=P107
LOC=P110
LOC=P113
LOC=P116
LOC=P123
OBUFT
IBUF
IOPAD
IFDE
David Summers
Naval Postgraduate School
TMR R3081 Project
Project: DCTRLVTR
Macro: DATA_C_IN
Date: 05/28/100



David Summers	Project: DCTRLVTR
Naval Postgraduate School	Macro: VOT_IN_OUT
TMR R3081 Project	Date: 05/28/100

LIST OF REFERENCES

1. Muolo, M. J., Space Handbook, An Analyst's Guide, vol 2, ch 1, pp. 1-21, Maxwell Air Force Base, AL, 1993.

2. National Semiconductor, Radiation Owner's Manual, National Semiconductor, 1999.

3. Weste, N. H. E. and Eshraghian, K., Principals of CMOS VLSI Design: A Systems Perspective, 2nd Edition, Addison-Wesley, Menlo Park, CA, 1994.

4. Asenek, V., et al., "SEU Induced Errors Observed in Microprocessor Systems," IEEE Transactions on Nuclear Science, Vol. 45, No. 6, December 1998. pp 2876-2883.

5. Underwood, C. I. and Oldfield, M. K., "Observed Radiation-Induced Degradation of Commercial-Off-The-Shelf (COTS) Devices Operating in Low-Earth Orbit," IEEE Transactions on Nuclear Science, Vol. 45, No. 6, December 1998. pp 2737-2744.

6. Johansson, R., "Two Error-Detecting and Correcting Circuits for Space Applications," Proceedings of Annual Symposium on Fault Tolerant Computing, 1996. pp 436-439.

7. Johnson, B. W., Design and Analysis of Fault Tolerant Digital Systems, Addison-Wesley, New York, NY, 1989.

8. Storey, N., Safety-Critical Computer Systems, Addison-Wesley, New York, NY, 1996.

9. Payne, Jr., J. C., "Fault Tolerant Computing Testbed: A Tool for the Analysis of Hardware and Software Fault Handling Techniques," Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1998.

10. Integrated Device Technology, Inc., The IDT79R3071, IDT79R3081 RISController Hardware User's Manual, Revision 2, Santa Clara, CA, 1994.

11. Ng, A., "AN-86, IDT79R3051 System Design Example," Online, Internet, Available: http://www.idt.com/docs/79R3051_AN_42226.pdf.

12. Groening, S. E., and Whitehouse. K. D., "Application of Fault-Tolerant Computing for Spacecraft Using Commercial-Off-The-Shelf Microprocessors," Master's Thesis, Naval Postgraduate School, Monterey, CA, June 2000.

13. Integrated Device Technology, Inc., "IDT79R3081 RISController with FPA," Online, Internet, Available: http://www.idt.com/docs/79R3081_DS_3483.pdf

14. Texas Instruments, Inc., "SN54AHCT540, SN74AHCT540 Octal Buffers/Drivers with 3-State Outputs," Online, Internet, Available: http://www-s.ti.com/sc/psheets/scls268j/scls268j.pdf

15. Texas Instruments, Inc., "TL7702A, TL7705A, TL7709A, TL7712A, TL7715A Supply-Voltage Supervisors," Online, Internet, Available: http://www-s.ti.com/sc/psheets/slvs028e/slvs028e.pdf

16. Texas Instruments, Inc., "CY54/74FCT373T, CY54/74FCT573T 8-Bit Latches," Online, Internet, Available: http://www-s.ti.com/sc/psheets/sccs021/sccs021.pdf

17. Texas Instruments, Inc., "SN54AHCT573, SN74AHCT573 Octal Transparent D-Type Latches with 3-State Outputs," Online, Internet, Available: http://www-s.ti.com/sc/psheets/scls243l/scls243l.pdf

18. Advanced Micro Devices, Inc., "Am27C010 1 Megabit (128K x 8-Bit) CMOS EPROM," Online, Internet, Available: http://www.amd.com/products/nvd/techdocs/10205.pdf

19. Wind River Systems, Inc., Tornado User's Guide (Windows Version), 1st ed., 1999.

20. Integrated Device Technologies, Inc., "IDT71024: CMOS Static RAM 1 Meg (128K x 8-Bit)," Online, Internet, Available: http://www.idt.com/docs/71024_DS_10380.pdf

21. Texas Instruments, Inc., "Design Notes: Interface Circuits for TIA/EIA-232-F," Online, Internet, Available: http://www-s.ti.com/sc/psheets/slla037/slla037.pdf

22. Texas Instruments, Inc., "TL16C750 Asynchronous Communications Element with 64-Byte FIFOs and Autoflow Control," Online, Internet, Available: http://www-s.ti.com/sc/psheets/slls191c/slls191c.pdf

23. Texas Instruments, Inc., "TL16C550A Asynchronous Communications Element," Online, Internet, Available: http://www-s.ti.com/sc/psheets/slls057d/slls057d.pdf

24. Maxim Integrated Products, "MAXIM +5V-Powered, Multichannel RS-232 Drivers/Receivers," Online, Internet, Available: http://pdfserv.maxim-ic.com/arpdf/1798.pdf

25. Integrated Device Technologies, Inc., "IDT72420, IDT72200, IDT72210, IDT72220, IDT72230, IDT72240: CMOS SyncFIFO 64 x 8, 256 x 8, 512 x 8, 1024 x 8, 2048 x 8, and 4096 x 8," Online, Internet, Available: http://www.idt.com/docs/72240_DS_1319.pdf

26. Atmel Corporation, "High Performance EE PLD: ATF22V10C," Online, Internet, Available: http://www.atmel.com/atmel/acrobat/doc0735.pdf

27. Logical Devices, Inc., CUPL: Universal Complier for Programmable Logic User Guide, 1991.

28. Wakerly, J. F., Digital Design: Principals and Practices, 3rd ed., Prentice Hall, Upper Saddle River, NJ, 2000.

29. Xilinx, Foundation Series Users Guide, Foundation Series Software, CD-ROM, Prentice Hall, Upper Saddle River, NJ, 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library..2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Road
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center 2
   MCCDC, Code C46
   2040 Broadway Street
   Quantico, VA 22134-5027

5. Marine Corps Representative 1
   Naval Postgraduate School
   Code 037, Bldg. 330 Ingersoll Hall, Room 116
   555 Dyer Road
   Monterey, CA 93943

6. Marine Corps Tactical Systems Support Activity 1
   Technical Advisory Branch
   Attn: Librarian
   Box 555171
   Camp Pendleton, CA 92055-5080

7. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

8. Professor Herschel Loomis Code EC/Lm..2
   Naval Postgraduate School
   Monterey, CA 93943-5121

9. Professor Alan Ross Code SP/Ra..2
Naval Postgraduate School
Monterey, CA 93943-5110

10. Captain David C. Summers, USMC 2
300 Ammunition Avenue
Odenton, MD 21113

11. LT John C. Payne, Jr., USN 1
6408 Chapel View Road
Clifton, VA 22024

12. Charlan Bonebright 1
5462 Mountainview Drive
Florence, MT 59833